General-Purpose Optimization Through Information Maximization

Alan J. Lockett

Technical Report AI 12-11 May 2012

`alan.lockett@utexas.edu`
`http://www.cs.utexas.edu/users/nn`

Artificial Intelligence Laboratory
Department of Computer Sciences
University of Texas
Austin, TX 78712

Copyright
by
Alan Justin Lockett
2012

The Dissertation Committee for Alan Justin Lockett
certifies that this is the approved version of the following dissertation:

General-Purpose Optimization through Information Maximization

Committee:

Risto Miikkulainen, Supervisor

Joydeep Ghosh

Raymond Mooney

Pradeep Ravikumar

Gordan Žitkovič

General-Purpose Optimization through Information Maximization

by

Alan Justin Lockett, B.A.; M.A.; M.S.C.S.

DISSERTATION

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
May 2012

Dedicated to my wife Alliene, who patiently and devotedly endured her husband's late night dalliance with abstract mathematics.

Acknowledgments

I wish to thank: My wife, Alliene, who showed undying devotion to an absentee husband and suffered many long evenings and weekends caring for three beautiful but demanding children by herself. My parents, who instilled in me a love of books and a belief that I could accomplish whatever I would set my mind to. My friends and family, who have supported and encouraged me in the difficult times. I would also like to thank: Prof. Wm Roger Louis, the distinguished historian of the British empire who encouraged me to pursue the Ph.D. in History; Prof. Clement Henry, who supervised my master's report in Turkish politics, and who supported me in more ways than one during an uncertain period of life; Dr. Paula de Witte, who was convinced that I was either crazy or a genius, and probably both, and who never ceased to believe in my potential; Prof. Victor Raskin, the Russian Jewish scholar of Linguistics, whose affable ways are legendary, and who took the time to argue the case for my admission into the Ph.D. program personally; Profs. Bruce Porter, Lorenzo Alvisi, and William Cook, who kindly and graciously supported my bid to enter Ph.D. program, and to whom I am indebted; my Ph.D. committee, who will have endured two theses wrapped in one; and last but not least, Prof. Risto Miikkulainen, who deserves praise beyond what I can give for his careful encouragement, unflagging optimism, valuable criticisms, and unending enthusiasm.

General-Purpose Optimization through Information Maximization

Publication No. ____________________

Alan Justin Lockett, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Risto Miikkulainen

The primary goal of artificial intelligence research is to develop a machine capable of learning to solve disparate real-world tasks autonomously, without relying on specialized problem-specific inputs. This dissertation suggests that such machines are realistic: If No Free Lunch theorems were to apply to all real-world problems, then the world would be utterly unpredictable. In response, the dissertation proposes the information-maximization principle, which claims that the optimal optimization methods make the best use of the information available to them. This principle results in a new algorithm, evolutionary annealing, which is shown to perform well especially in challenging problems with irregular structure.

Table of Contents

Acknowledgments **vi**

Abstract **vii**

List of Tables **xvii**

List of Figures **xxii**

Chapter 1. Introduction **1**

1.1 Motivation 1
1.2 The Optimization Task 3
1.3 Spaces of Optimizers 6
1.4 Optimizer Performance and No Free Lunch 11
1.5 Information-Maximizing Optimization 15
1.6 Guide to the Reader 17
1.7 Conclusion 19

Chapter 2. Review of Optimization Methods **21**

2.1 Overview 21
2.2 Gradient Methods 23
  2.2.1 Early Methods 23
  2.2.2 Newton's Method 24
  2.2.3 Gradient Descent 26
  2.2.4 Line Search 27
  2.2.5 Gradient Descent in Parametric Models 28
2.3 Problems with Gradient Methods 29
2.4 Direct Search 33
  2.4.1 Simplicial Methods 34
  2.4.2 Generating Set Search 34

2.5 Stochastic Optimization 36
  2.5.1 Simulated Annealing 36
  2.5.2 Justification for Stochastic Methods 37
2.6 Evolutionary Methods 39
  2.6.1 Overview 39
  2.6.2 Genetic Operators 42
  2.6.3 Random Heuristic Search 44
2.7 Quasi-Evolutionary Methods 45
  2.7.1 Differential Evolution 46
  2.7.2 Estimation of Distribution Algorithms 47
  2.7.3 Natural Evolution Strategies 49
2.8 Conclusion 50

Chapter 3. Functional Analysis of Optimization **52**
3.1 Motivation 52
3.2 Optimizer Space 53
  3.2.1 Assumptions 54
  3.2.2 Basic Representation 55
  3.2.3 Notation and Conventions 58
  3.2.4 Information Restrictions 60
  3.2.5 Computability of Optimizers 62
3.3 Algebraic Operations 65
  3.3.1 Convolution 65
  3.3.2 Trajectory Truncation 67
  3.3.3 Convex Combination 68
3.4 Measure-Valued Operators: a Normed Vector Space 69
  3.4.1 The Normed Vector Space $\mathcal{MF}$ 70
  3.4.2 Vector Subspaces of $\mathcal{MF}$ 73
3.5 Conclusion 75

Chapter 4. A Unified View of Population-Based Optimizers 77
4.1 Population-Based Optimizers 77
4.1.1 Motivation and Goals 77
4.1.2 Formalities 78
4.1.3 Constructive Operators 80
4.1.4 Examples: Random Walkers 82
4.1.5 Example: Simulated Annealing 83
4.2 Evolutionary Algorithms 86
4.2.1 Characteristics of an Evolutionary Algorithm 86
4.2.2 Selection, Recombination, and Mutation 87
4.2.3 Genetic Algorithms 92
4.2.4 Evolution Strategies 97
4.2.5 The $(1+1)$–ES as the Norm Limit of Simulated Annealing 100
4.3 Quasi-Evolutionary Algorithms 103
4.3.1 Particle Swarm Optimization 103
4.3.2 Differential Evolution 105
4.3.3 Parameterized Quasi-Evolutionary Methods 107
4.3.4 Non-Quasi-Evolutionary Algorithms 108
4.4 Conclusion 110

Chapter 5. Continuity of Optimizers 111
5.1 Background and Motivation 111
5.2 Deterministic Optimizers 115
5.3 Evolutionary Algorithms 116
5.3.1 Continuity of Convolution 116
5.3.2 Sample Convergence and Continuity 119
5.3.3 Sample Divergence and Discontinuity 125
5.3.4 Discontinuities of Specific Selection Rules 127
5.4 Quasi-Evolutionary Algorithms 130
5.4.1 Parameterized Methods 131
5.4.2 Differential Evolution and Locally Improving Optimizers 132
5.5 Stochastic Gradient Descent 134
5.6 Conclusion 135

Chapter 6. The Optimization Process **137**
6.1 Construction of the Optimization Process 137
6.2 The Space of Long-Running Optimizers 140
6.3 Increasing Information and Stopping Times 142
  6.3.1 Filtrations and Information 142
  6.3.2 Stopping Times 143
6.4 Continuity of the Optimization Process 145
6.5 Conclusion 151

Chapter 7. Performance Analysis **152**
7.1 Performance Criteria 152
  7.1.1 Evaluation by Average Error 154
  7.1.2 Evaluation by Hitting Time 155
  7.1.3 Evaluation by Success Probability 156
  7.1.4 Evaluation by Error at a Stopping Time 157
7.2 Properties of Performance Criteria 159
  7.2.1 Nonlinearity 159
  7.2.2 Progressive Decomposability 160
  7.2.3 Dependence on the Error Sequence 163
7.3 Continuity of Performance 164
  7.3.1 Continuity In Objectives 165
  7.3.2 Continuity in Optimizers 169
  7.3.3 Sample Convergence and Performance Continuity 171
7.4 Conclusion 174

Chapter 8. Performance Experiments **175**
8.1 Experimental Setup 175
  8.1.1 Benchmarks 176
  8.1.2 Algorithms 178
  8.1.3 Scaling Factors 179
8.2 Experimental Results 181
8.3 Restarting to Bootstrap Performance 188
8.4 Illustrating Continuity of Performance 191

8.4.1 Continuity in Objectives 192
8.4.2 Continuity in Optimizers 192
8.4.3 An Example of Discontinuity 194
8.5 Principal Components Analysis 198
8.6 Conclusion 204

Chapter 9. No Free Lunch Does Not Prevent General Optimization 205
9.1 Overview of No Free Lunch 205
9.1.1 NFL Basics 206
9.1.2 Function Priors and Closure Under Permutation 207
9.1.3 Infinite Extensions of NFL 209
9.2 NFL Preliminaries 210
9.2.1 Function Priors and Random Objectives 211
9.2.2 NFL Priors 212
9.2.3 Properties of Function Priors 216
9.3 NFL Theorems 218
9.3.1 Implications of Strong and Weak NFL 218
9.3.2 NFL Implications on the Evaluation Path 220
9.3.3 Sufficient Conditions for NFL 225
9.3.4 NFL Identification Theorem 228
9.4 Conclusion 229

Chapter 10. The Geometry of Optimization and the Optimization Game 231
10.1 The Reality Prior 231
10.1.1 The Frequency of NFL Priors 232
10.1.2 Diffusion Prior 233
10.1.3 The Universal Prior 236
10.1.4 The Meaning of NFL 237
10.2 Duality and the Geometry of Optimization 238
10.2.1 Duality Based on Average Error 238
10.2.2 Duality Under Arbitrary Performance Criteria 241

10.2.3 Duality and Information Restrictions 242
10.3 Linear Functionals 244
10.3.1 Continuous Linear Functionals 244
10.3.2 Performance-improving Linear Extensions 245
10.3.3 Performance-based Linear Projections 246
10.4 The Optimization Game 247
10.4.1 Game Theory and Optimization 247
10.4.2 The Role of Information 249
10.4.3 The Objective Player Wins 250
10.4.4 Optimal Optimization through Information-Maximization 252
10.4.5 Martingale Optimization 254
10.4.6 Curiosity Search 256
10.4.7 Strategic Forgetting 257
10.5 Conclusion 258

Chapter 11. The Evolutionary Annealing Method 260
11.1 Foundation 260
11.1.1 Martingales vs. Markov Chains 261
11.1.2 Characteristics of Evolutionary Annealing 261
11.1.3 Expected Proportional Selection 262
11.2 The Evolutionary Annealing Approach 265
11.2.1 Formal Context and Assumptions 265
11.2.2 Basic Algorithm 265
11.2.3 Partitioning the Space 269
11.2.4 Annealed Tournament Selection 271
11.3 Convergence Properties 273
11.3.1 Convergence Proof 273
11.3.2 Convergence Rates 284
11.4 Conclusion 286

Chapter 12. Evolutionary Annealing In Euclidean Space 287
12.1 Evolutionary Annealing in Euclidean Space 287
12.1.1 Instantiation Details 287
12.1.2 Sampling Algorithms for Annealed Selection 288
12.1.3 Sampling Annealed Tournament Selection 291
12.1.4 Implementation 293
12.2 Experiments with REA 294
12.2.1 Experimental Results 295
12.2.2 Analysis of REA Results 305
12.3 Problem Alignment 309
12.4 Conclusion 313

Chapter 13. Neuroannealing 314
13.1 Evolving Recurrent Neural Networks 314
13.1.1 RNN Basics 315
13.1.2 Neuroevolution 316
13.1.3 Evaluating the Performance of NEAT 318
13.1.4 Experimental Hypothesis 319
13.2 Evolutionary Annealing for Neural Networks 320
13.2.1 Layered RNNs 320
13.2.2 Base Measure for RNNs 321
13.2.3 Redundant Representations 324
13.2.4 Partitioning Networks 324
13.2.5 Network Mutations 327
13.2.6 Neuroannealing Instantiation 330
13.3 Neuroannealing Experiments 331
13.3.1 Experimental Setup 331
13.3.2 Double Pole-Balancing 332
13.3.3 Multiplexers 335
13.3.4 Concentric Spirals 338
13.3.5 Currency Trading 342
13.4 Neuroannealing Discussion and Future Work 346
13.5 Conclusion 348

Chapter 14. Disscussion and Future Work 349
14.1 Formalization of Optimizers 349
14.1.1 Dynamic and Stochastic Objectives 349
14.1.2 Alternative Characterizations 351
14.1.3 Convex Control of Optimizer Portfolios 353
14.1.4 Formalization Conclusion 356
14.2 General-Purpose Optimizers 356
14.2.1 Extending NFL to Information-Restricted Optimizers . 356
14.2.2 General-Purpose Optimization Conclusion 358
14.3 Martingale Optimization 359
14.3.1 Proving the Optimality of Information Maximization . . 359
14.3.2 Semantic Partitioning 360
14.3.3 Applications to Other Search Domains 360
14.3.4 Information Compression 361
14.3.5 Information Maximization in Stochastic and Dynamic Environments 361
14.3.6 Information Maximization Conclusion 362
14.4 Conclusion 362

Chapter 15. Conclusion 364
15.1 Contributions 364
15.1.1 Significance of the Formal Approach 364
15.1.2 Significance of the NFL Identification Theorems 366
15.1.3 Significance of Information Maximization 368
15.2 Final Thoughts 369

Appendices 373

Appendix A. Performance Experiment Results 374

Appendix B. Automated Currency Exchange Trading 426
B.1 Currency Trading Background 426
B.2 Currency Trading Experiments 429
B.2.1 Experimental Setup 429
B.2.2 Currency Exchange Rate Data Set 431

Bibliography **433**

Vita **454**

List of Tables

8.1 Benchmarks for Experimental Validation with dimension $d = 5, 10, 25$. Minimum for $d = 5$. 177

11.1 Table of symbols relating to the convergence proof 274

12.1 Performance statistics for Evolutionary Annealing on a 2GHz Intel Core 2 Duo processor using the open-source implementation available at http://pypi.python.org/pypi/PyEC. For each number of observed points, the table gives the time in milliseconds for sampling one point, for inserting one point into the partition tree, for inserting one point into the ranked score tree, and for the total processing overhead per function evaluation. Complexity grows logarithmically in the number of points. . . 295

12.2 Learning rates η for REA-P and REA-T tested in the experiments. Lower values yield higher success probability at the cost of slower convergence. 303

13.1 Published results for selected methods on both versions of the Double Pole-Balancing task, as given by Gomez et al [80]. Reported quantity is the average number of evaluations before success, with failed trials excluded (i.e., $\hat{\psi}_0^N$, $N = 100,000$ for new results). Results for neuroannealing are new (as indicated by the asterisks), as well as the results for NEAT (determined experimentally using the parameters published by Stanley [197]).333

13.2 Results of neural network experiments on both versions of the double pole-balancing task using the performance criteria of Chapter 8: success probability (σ_ϵ^N), hitting time on success ($\hat{\psi}_\epsilon^N$), final error (ζ_T), average error (ϕ_1), and weighted average error (ϕ_2). The first standard deviation is provided where possible; the value for σ_ϵ^N is accurate within ± 0.005 with $p < 0.05$. The error threshold ϵ was chosen separately for each task and is given in the table. Neuroannealing succeeds on both tasks, but requires more objective evaluations than NEAT. 334

13.3 Results of neural network experiments on the multiplexer problem using the performance criteria of Chapter 8. The error threshold ϵ for each task is given in the table. Neuroannealing outperforms NEAT on the multiplexer problems in terms of final error; these results are statistically significant ($p < 0.01$). 337

13.4 Results of neural network experiments on the concentric spirals problem using the performance criteria of Chapter 8. The error threshold ϵ is given in the table. Neuroannealing outperforms NEAT substantially in terms of final error; this result is statistically significant ($p < 0.01$). 341

13.5 Results of neural network experiments on the currency trading task using the performance criteria of Chapter 8. For currency trading, the global optimal value are unknown, and the values for σ_ϵ^N and $\hat{\psi}_\epsilon^N$ were found by averaging over trials on which the final account value was greater than 250. Neuroannealing and NEAT both perform well, but neuroannealing performs slightly better, achieving higher success probability, higher final account values, and the largest overall account value. 345

A.1 Scaling factors used for the scaled variants of ζ_{T_m}, ϕ_1, and ϕ_2 by dimension d. 375

A.2 Learning rates η for REA-P and REA-T tested in the experiments. Lower values yield higher success probability at the cost of slower convergence. 376

A.3 Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 5 dimensions. 377

A.4 Results on performance criterion σ_ϵ^N with $\epsilon = 0.010$ and $N = 250,000$ in 5 dimensions. 378

A.5 Results on performance criterion σ_ϵ^N with $\epsilon = 0.001$ and $N = 250,000$ in 5 dimensions. 379

A.6 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.100$ and $N = 250,000$ in 5 dimensions. 380

A.7 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.010$ and $N = 250,000$ in 5 dimensions. 381

A.8 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.001$ and $N = 250,000$ in 5 dimensions. 382

A.9 Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 5 dimensions. 383

A.10 Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 5 dimensions. 384
A.11 Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 5 dimensions. 385
A.12 Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 5 dimensions. 386
A.13 Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 5 dimensions. 387
A.14 Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 5 dimensions. 388
A.15 Results on performance criterion ϕ_1 (scaled) in 5 dimensions. . 389
A.16 Variance for performance criterion ϕ_1 (scaled) in 5 dimensions. 390
A.17 Results on performance criterion ϕ_2 (scaled) in 5 dimensions. . 391
A.18 Variance for performance criterion ϕ_2 (scaled) in 5 dimensions. 392
A.19 Results on performance criterion σ_ϵ^N with $\epsilon = 1.000$ and $N = 250,000$ in 10 dimensions. 393
A.20 Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 10 dimensions. 394
A.21 Results on performance criterion σ_ϵ^N with $\epsilon = 0.010$ and $N = 250,000$ in 10 dimensions. 395
A.22 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 1.000$ and $N = 250,000$ in 10 dimensions. 396
A.23 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.100$ and $N = 250,000$ in 10 dimensions. 397
A.24 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.010$ and $N = 250,000$ in 10 dimensions. 398
A.25 Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 10 dimensions. 399
A.26 Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 10 dimensions. 400
A.27 Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 10 dimensions. 401
A.28 Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 10 dimensions. 402
A.29 Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 10 dimensions. 403

A.30 Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 10 dimensions. 404

A.31 Results on performance criterion ϕ_1 (scaled) in 10 dimensions. 405

A.32 Variance for performance criterion ϕ_1 (scaled) in 10 dimensions. 406

A.33 Results on performance criterion ϕ_2 (scaled) in 10 dimensions. 407

A.34 Variance for performance criterion ϕ_2 (scaled) in 10 dimensions. 408

A.35 Results on performance criterion σ_ϵ^N with $\epsilon = 10.000$ and $N = 250,000$ in 25 dimensions. 409

A.36 Results on performance criterion σ_ϵ^N with $\epsilon = 1.000$ and $N = 250,000$ in 25 dimensions. 410

A.37 Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 25 dimensions. 411

A.38 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 10.000$ and $N = 250,000$ in 25 dimensions. 412

A.39 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 1.000$ and $N = 250,000$ in 25 dimensions. 413

A.40 Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.100$ and $N = 250,000$ in 25 dimensions. 414

A.41 Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 25 dimensions. 415

A.42 Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 25 dimensions. 416

A.43 Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 25 dimensions. 417

A.44 Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 25 dimensions. 418

A.45 Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 25 dimensions. 419

A.46 Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 25 dimensions. 420

A.47 Results on performance criterion ϕ_1 (scaled) in 25 dimensions. 421

A.48 Variance for performance criterion ϕ_1 (scaled) in 25 dimensions. 422

A.49 Results on performance criterion ϕ_2 (scaled) in 25 dimensions. 423

A.50 Variance for performance criterion ϕ_2 (scaled) in 25 dimensions. 424

A.51 Results of neural network experiments using the performance criteria of Chapter 8: success probability (σ_ϵ^N), hitting time on success ($\hat{\psi}_\epsilon^N$), final error (ζ_T), average error (ϕ_1), and weighted average error (ϕ_2). The first standard deviation is provided where possible; the value for σ_ϵ^N is accurate within ± 0.005 with $p < 0.05$. The error threshold ϵ was chosen separately for each task and is given in the table. For Currency Trading, raw fitness values are shown instead of errors. Neuroannealing outperforms NEAT on the Multiplexers and Concentric Spirals. 425

List of Figures

1.1 Three example objective functions. Figure 1.1(a) is *unimodal* and possesses a unique minimum, easily located analytically or by gradient methods. Figure 1.1(b) is *multimodal* but *periodic*. Gradient methods will fail, but the periodicity can be used to locate the optimum. Figure 1.1(c) is multimodal and irregularly structured. Such problems can be difficult to solve, particularly in high dimension. 6

2.1 (a)Attraction basins for a sum of ten Gaussian kernels under conjugate gradient descent on a square region. Plots were generated by sampling a 300×300 grid to discover which local maximum results from each grid point using conjugate gradient descent. Different colors indicate regions that result in different local maxima. Contour lines are included in gray to indicate the location of the Gaussians. The global maximum is the red region in the upper left. Its area among possible starting points is small; only 639 of the $90,000$ sample points (0.7%) converge to the global maximum. (b) A bar chart showing the probability of arriving at the best local maxima, ordered by quality from left to right. Again, gradient descent is unlikely to find the best solution. 32

8.1 Heat maps for the twelve benchmark functions in two dimensions ($d = 2$). The benchmarks *whitley* and *griewank* are scaled to show the critical region. These benchmarks include unimodal, multimodal, periodic, irregular, and discontinuous functions, resulting in a broad test of an optimizer's capabilities . 180

8.2 Performance values for selected optimizers on the twelve benchmarks using the performance criterion ϕ_1, which averages global error over $250,000$ evaluations, starting after $10,000$ evaluations. Lower values are better. DE, CMA-ES, CMA-ES-R, GSS-R, and NM-R perform best on this performance criterion. 183

8.3 Performance values for selected optimizers on the twelve benchmarks using the performance criterion ϕ_2, which sums global error with an exponential decay over $250,000$ evaluations, starting after $10,000$ evaluations. Lower values are better. Unlike ϕ_1, ϕ_2 counts earlier errors more heavily, and thus prefers optimizers that converge faster, such as NM, GSS, and CMA-ES. 184

8.4 Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\zeta_{T_{250,000}}$, which reports the minimum global error after $250,000$ unique evaluations. Lower values are better. Because it does not sum over multiple time steps, ζ_T communicates little information about the convergence rate. DE performs best among the optimizers, with CMA-ES close behind. Restarting improves performance with enough evaluations, and so CMA-ES-R, GSS-R, NM-R, and CG-R each perform well. 186

8.5 Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\frac{1}{100}\psi_{\epsilon}^{N}$ with $\epsilon = 0.01$ and $N = 250,000$. This criterion records the average number of evaluations before the minimum global error drops below ϵ, capped at a maximum of N. Lower values are better. Scale as shown runs from zero to $2,500$ and represents the number of generations until the hitting time with a notional population size of 100. Because many trial runs fail on these benchmarks, the numbers are typically high, and often near $2,500$. Very low values, as seen for *sphere*, indicate fast convergence. In general, the success-only hitting time $\hat{\psi}_{\epsilon}^{N}$ is more preferable than ψ_{ϵ}^{N} for measuring performance. 187

8.6 Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.01$ and $N = 250,000$ (see Equation 7.8). This criterion records the average number of evaluations before the minimum global error drops below ϵ on trial runs where this error is attained. If the error threshold is never attained, the value is set at N. Lower values are better. Scale as shown runs from zero to $2,500$ and represents the number of generations until the hitting time with a notional population size of 100. These values give a clear picture of the relative convergence speed of various optimizers. For example, on successful trials, CMA-ES, GSS, and NM converge very quickly, whereas DE converges, but more slowly. 189

8.7 Performance values for selected optimizers on the twelve benchmarks using the performance criterion σ_{ϵ}^{N} with $\epsilon = 0.01$ and $N = 250,000$. This criterion computes the probability of attaining global error within ϵ of the true optimum. Values run from zero to one; higher values are better. Overall, CMA-ES (especially CMA-ES-R) performs best on this performance criterion. DE, GSS-R, and NM-R also perform well on different objectives. 190

8.8 Change in performance by NM-R as the objective changes smoothly from *langerman* ($\alpha = 0$) to *shekel* ($\alpha = 1$). The x-axis ranges over values of α, the y-axis over performance values. The first standard deviation is also plotted on either side of the performance where possible. Panels show the performance criteria ϕ_1, ζ_T, $\hat{\psi}^N_\epsilon$, and σ^N_ϵ, respectively. As predicted by the theory, performance on these optimizers changes smoothly and nonlinearly as a function of the objective. 193

8.9 Change in performance as the optimizer changes smoothly from PSO with $\omega = -.5, \phi_g = \phi_p = 2$ ($\alpha = 0$) to DE with CR=.2, F=.2 ($\alpha = 1$). The x-axis ranges over values of α, the y-axis over performance values. The first standard deviation is also plotted on either side of the performance where possible. The panels show the performance criteria ϕ_1, ζ_T, $\hat{\psi}^N_\epsilon$, and σ^N_ϵ, respectively, with $\epsilon = 25$ for *schwefel*. As predicted by the theory, performance on these optimizers changes smoothly and nonlinearly as a function of the optimizer. Interestingly, at $\alpha = .95$, $\mathcal{G}_\alpha$ outperforms PSO and DE on ζ_T, although the result is not statistically significant. Convex combinations of algorithms were formally proposed in this dissertation, and Section 7.2.1 suggested that convex combinations may outperform pure algorithms. This example validates this conjecture and confirms the value of the formal approach in this dissertation. 195

8.10 Change in performance of rGA-2 and DE for zero-centered prisms of different heights in 25 dimensions, measured on t_α from Equation **??** for 21 values of α evenly spaced in the interval $[-0.1, 0.1]$. The x-axis ranges over values of α, the y-axis over performance values. Panels in the left column show the performance of rGA-2, and panels in the right column show the performance of DE. The rows show the performance criteria ϕ_1, ζ_T, $\hat{\psi}^N_\epsilon$, and σ^N_ϵ, respectively. DE has a discontinuity at t_0. 197

8.11 PCA plots for the test algorithms on the benchmarks. These plots show the proximity between various optimizers based on their performance on ϕ_1 and ζ_T. The resulting layout of optimizers reveals interesting new relationships among the algorithms. 200

8.12 The first three PCA components in the ϕ_1 basis, with CG and CG-R included. The first component separates conjugate gradient descent from the other methods, and the second component separates the restarted version from the non-restarted version. This plot shows that CG and CG-R are indeed distinct from the other methods in terms of performance on ϕ_1. 201

8.13 PCA plots for eleven algorithms on eleven benchmarks. These plots show the proximity between various optimizers based on their performance on $\hat{\psi}_{\epsilon}^{N}$ and σ_{ϵ}^{N}. The resulting layout of optimizers reveals interesting new relationships among the algorithms. 203

10.1 An example showing an estimated objective function such that the variance of the estimate increases with distance from the control points. The highest variance occurs at $x = 0.75$, but a new optimum is much more likely to be found near 2.25. When optimizing an objective function, merely reducing variance is not enough; the objective values must be taken into account as well. The optimal point to choose should trade off between reducing variance and improving the objective value. 255

11.1 Example run of evolutionary annealing on Shekel's Foxholes in two dimensions (shown in Figure 8.1(e)). Images are heat maps displaying the estimated probability density of evolutionary annealing, that is, the probability that each point will occur in the next generation of evolutionary annealing. White areas are more probable, and dark areas are less probable. Successive frames show how the probability density changes once 50, 125, 250, and 500 points have been evaluated. The resulting distribution increasingly models the fitness function; comparison with Figure 8.1(e) confirms that after 500 evaluations, evolutionary annealing has focused on the true global optimum. 263

11.2 Progression of partition regions $\{E_n^a\}$ on Shekel's Foxholes during the run of evolutionary annealing in Figure 11.1 in two dimensions. Images are heat maps displaying the selection probability of each region; light regions have a higher selection probability. Successive frames show how the partition regions gradually model the shape of the fitness function after 10, 25, 100, and 250 points have been evaluated. 271

12.1 Performance of REA, DE, and CMA-ES in five dimensions on the average unweighted error ϕ_1 (scaled), as reported in Figure 8.2 for all optimizers. Lower values are better. All four optimizers are generally comparable on this criterion, with REA-T performing best on *salomon*, *rosenbrock*, *shekel*, *langerman*, *whitley*, and *weierstrass*. REA-T generally has lower error than REA-P, although REA-P also performs well in five dimensions. 296

12.2 Performance of REA, DE, and CMA-ES in five dimensions on the average weighted error ϕ_2 (scaled), as reported in Figure 8.3 for all optimizers. Lower values are better. The criterion ϕ_2 emphasizes early errors, and thus favors faster converging optimizers such as CMA-ES. REA-T converges at the same rate as CMA-ES in several cases, and often makes less early errors than DE. REA-T is the best on *langerman* and *weierstrass* for ϕ_2. 297

12.3 Performance of REA, DE, and CMA-ES in five dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled), as reported in Figure 8.4 for all optimizers. Lower values are better. REA-T performs best on *schwefel*, *shekel*, *langerman*, *whitley*, and *weierstrass*, with lower average error at the end of evaluation. 298

12.4 Performance of REA, DE, and CMA-ES in five dimensions on the success probability σ_ϵ^N with $\epsilon = 0.01$ and $N = 250,000$, as reported in Figure 8.7 for all optimizers. Higher values are better. REA-T generally performs best on the irregular problems, such as *shekel*, *langerman*, and *whitley*. 299

12.5 Performance of REA, DE, and CMA-ES in five dimensions on the average hitting time $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.01$ and $N = 250,000$, as reported in Figure 8.6 for all optimizers. Lower values are better; 2500 is maximum value. REA-T converges at about the same rate as CMA-ES when successful, suggesting that its performance could be boosted by restarting. In general, REA-T is more accurate than CMA-ES. 300

12.6 Performance of REA, DE, and CMA-ES in 10 dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled). The performance of REA-P degrades as the dimension increases. REA-T performs best on several problems, including *salomon*, *shekel*, *langerman*, and *weierstrass*. 301

12.7 Performance of REA, DE, and CMA-ES in 25 dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled). REA-P performed substantially worse and was omitted. All algorithms perform relatively well on *sphere*, *griewank*, *rosenbrock*, and *whitley*. REA-T performs best among these three methods on *weierstrass*. 302

12.8 Success probabilities for REA-T on selected benchmarks in five dimensions for four different learning rates. Decreasing the learning rate improves the success probability overall but requires more fitness evaluations. 306

12.9 Average error rates for REA-T on selected benchmarks in five dimensions for four different learning rates. The black solid line is the average error for the largest learning rate in Table 12.2; the grey solid line is the second largest; the black dotted line is the third largest, and the grey dotted line is the smallest learning rate. Decreasing the learning rate thus reduces error overall at the cost of increased error in early generations. . . . 306

12.10 Examples of priors drawn from G_{25}, G_{50}, and G_{75} using Equation 12.12. The random objective G_∞ may be well aligned with REA, supposing a fixed, decaying variance was used in place of REA's area-sensitive variance decay. An area-sensitive decay would be more smooth, particularly in sparse regions. The fractal structure of G_n is clearly evident in these images. This function roughly resembles *langerman*, on which REA-T performs best among all optimizers tested, suggesting that REA is well aligned with this random objective. 312

13.1 The multiplexer learning problem. The correct output is determined by the value of the data input at the address specified. With three address bits (Mux35 and Mux36), not all addresses were used to simplify the problem for a neural network. 336

13.2 Illustration of the Concentric Spirals Problem, in which points must be correctly classified as belonging to interlaced black and white spirals. The left panel shows the 197 control points used to define the problem, and the right panel shows the state space divided according to whether a black or white point is closer. The percentage of correct classifications on the points in the 100 × 100 grid in the right panel was used for training neural networks. 340

13.3 State space classification for the concentric spirals problem as learned by neuroannealing. Objective values and network sizes are shown for each solution. More accurate solutions require larger networks. Neuroannealing is able to discover these solutions, whereas NEAT does not. 343

Chapter 1

Introduction

The discovery of an effective general-purpose learning algorithm is the Holy Grail of artificial intelligence research. Such an algorithm would be capable of learning to solve disparate real-world tasks autonomously, without relying on specialized problem-specific inputs. This quest has come to be regarded as quixotic in light of the No Free Lunch theorems, which proved that, when averaged over all problems, every learning method performs equally. This sentiment reflects a misunderstanding of the meaning of No Free Lunch, which only applies in restricted circumstances. This dissertation establishes a formal context for studying optimization within which it is proven that if No Free Lunch did apply to all real-world problems, then the world would be utterly unpredictable. Predictable physical laws could not exist, neither gravity, nor electromagnetism, nor indeed the atom itself. Regularity would be outlawed, and reality would be reduced to an absurd sequence of random events. In fact, real phenomena are relatively predictable on small time scales. Thus there must exist learning strategies that outperform others on the general class of real-world problems. In response, this dissertation proposes the information-maximization principle, which claims that the optimal optimization methods in any problem setting are those that make the best use of the information available to them. This principle is applied to develop a new algorithm, evolutionary annealing, which solves optimization tasks in spaces of real vectors and neural networks.

1.1 Motivation

Nearly every problem to which human ingenuity is applied either consists of or contains an optimization task. When constructing a building, one wishes to use the least amount of materials and labor to achieve certain standards of size, quality, and content. When planning a trip, one seeks a route of transportation that minimizes distance and maximizes speed. When designing the layout for an integrated circuit, one desires an arrangement of components that minimizes both fabrication errors and surface area. In machine learning research, optimization is ubiquitous, with nearly every learning task being formalized as an optimization problem in which errors are to be minimized and performance is to be maximized. In this setting, learning and optimization are closely intertwined, and a general-purpose optimization method is required in order to implement a general-purpose learner.

This dissertation studies methods for solving optimization tasks in the abstract. Previous studies have examined the optimization task in the context of a particular method or class of methods. The present study encompasses all possible iterative optimization methods by analyzing the sequence of proposed optima generated by each method. In this manner, optimization methods are identified as mathematical objects in a normed vector space with well-defined notions of distance, continuity, and limits. This text focuses primarily on *trajectory-restricted* optimization methods, that is, methods that propose new solutions based solely on the measured objective value of previously proposed solutions, as opposed to methods that use additional information, such as gradient evaluations. The formalization pertains to all optimizers, however, and some results will be presented for larger classes of optimizers as well.

This dissertation makes four main contributions. First, the particular formalization of optimizers offered here is of high value in itself as a context within which broadly applicable theorems can be stated and proven. Prior formalizations have been limited to finite spaces [162, 218] or subsets of optimizers [111, 206]. Secondly, an explicit account is given of how the performance of an optimizer can be defined and measured objectively. Many standard metrics for performance exist currently, but there has previously been no means of studying their analytic properties. Thirdly, the No Free Lunch theorems for optimization [162, 171, 218] are extended to arbitrary measure spaces and the exact conditions under which such theorems hold are expounded for the first time. Specifically, it is shown that No Free Lunch only holds if objectives are assumed to be unlearnable a priori. Since the human experience in particular demonstrates that general learning does occur, it follows that general-purpose optimization is possible for real-world problems. Fourthly and finally, the role of information in optimization is discussed at length, and evolutionary annealing is proposed as an optimization method that makes full use of information and is experimentally effective on several optimization problems.

This dissertation presents the mathematical theory of stochastic global optimization on arbitrary measure spaces with static objective functions. The material is quite broad and covers a wide scope of information. This text presents the first results of this potentially powerful approach. There are a substantial number of questions left open and a large quantity of obvious consequences that might be added to this body of work. Some possible extensions are presented in Chapter 14. This dissertation aims to present a comprehensive and clear account of the relevant aspects of optimization methods, with a focus on the analysis of optimizer performance.

In the remainder of this chapter, the optimization task is discussed in detail and the formal context adopted in this text is motivated. The implication of the results obtained in later chapters is previewed, and a guide to the content of the dissertation is provided.

1.2 The Optimization Task

At its core, an optimization task consists of a configurable system, a set of quantifiable objectives for the system, and potentially a set of observable environmental factors. A solution to an optimization problem prescribes an input configuration (possibly as a function of the observable environment) such that the objectives attain their maximal or minimal values. The set of admissible configurations is the *search space* or the *search domain.* The objectives for the system are considered as functions taking system configurations as input and producing the objective values as output. As such, in academic settings, optimization is almost always studied in terms of finding the minimal or maximal values of a computable function. The function is referred to as the *objective function* or the *fitness function* depending on the context. In the presence of multiple objectives, the objectives may be combined into a single metric, or each objective can be treated independently. The latter case is termed is termed *multi-objective optimization* and is not discussed further in this text.

Formally, let X be the search domain, and suppose the objective function f is some real-valued function over X, i.e. $f \in \mathbb{R}^X$. Optimization is formally considered to mean minimization, since a function f can be maximized by minimizing $-f$. It is then further assumed that $\inf_{x \in X} f(x) > -\infty$. Then the goal of the optimization task is to find one or more $x^* \in X$ such that f takes its minimal value on x^*, i.e.

$$f(x^*) = \inf_{x \in X} f(x). \tag{1.1}$$

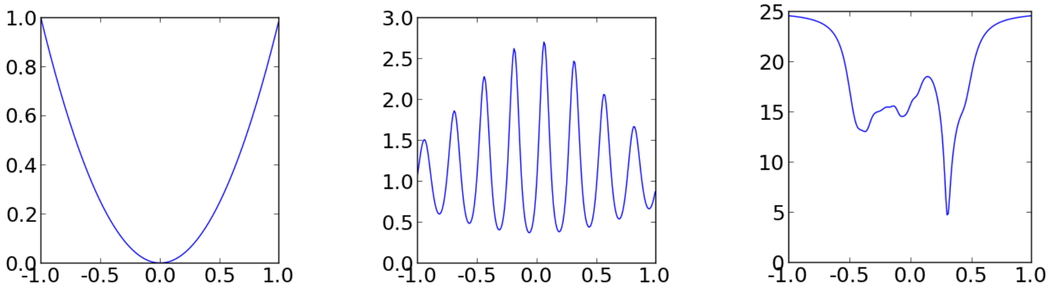
Many times, a general search space such as $\mathbb{R}$ is provided along with a set of constraints that defines a search domain $C \subseteq \mathbb{R}$, and a minimal value for f is sought from among the elements of the set C. This setting is termed *constrained optimization.* This text focuses on *unconstrained optimization*, although for generality, the constraints can be assumed to be built into the space, so that $X = C$ from the discussion above.

A simple example is in order. Consider the task of driving a car from a given location to a second fixed location through an urban setting. The car is the system, and its relevant configurable parts consist of the steering wheel, the gas pedal, and the brake. The car can be controlled by specifying at each point in time the rotational force on the steering wheel and the downward pressure on the gas pedal and the brake. In this case, the proper controls for the car depend on the state of the environment. The task of driving a car has several objectives. First, the car must arrive at the correct destination as quickly as possible. Secondly, the car must obey traffic laws, remaining within the appropriate lanes, stopping at red lights, maintaining appropriate speed, and signaling turns in advance. Thirdly, the car must avoid collisions with other vehicles, pedestrians, animals, and objects. A solution to the driving task specifies the force on the steering wheel, brakes, and accelerator as a function of the observed environmental state. The objective function in this case is typically a simulation environment or a real-world test in which the car is controlled by a proposed solution and its performance is measured in terms of the three main criteria above. The search domain is a space of functions that map the observations of the environment to driving decisions. A good solution safely drives the car from its starting point to its destination along an efficient route while following the traffic laws and avoiding collisions.

The example above can be mapped into a formal optimization problem by identifying the car's sensors as an array of m real numbers in $\mathbb{R}^m$ and the three controls as an element of $\mathbb{R}^3$. The search domain X consists of all functions from $\mathbb{R}^m$ to $\mathbb{R}^3$. For any controller $x \in X$, one may define three objectives f_1, f_2, and f_3, such that $f_1(x)$ records the distance from the desired destination at the end of the simulation, $f_2(x)$ counts the number of traffic law violations, and $f_3(x)$ indicates the risk of a collision or other catastrophic mistake over the course of a simulation run. Then a suitable objective function would be $f = \sum_{i=1}^{3} \alpha_i f_i$ with $\alpha_i > 0$, where the α_i balance the importance of each objective. A solution to this optimization task would output a controller x^* for the vehicle such $f(x^*)$ is minimal, that is, such that the vehicle reaches its destination while obeying traffic laws and avoiding collisions.

An iterative optimization method or *optimizer* proposes a sequence of potential solutions to an optimization task, $x_1, x_2, x_3, \dots \subseteq X$. The quality of the solutions should increase as the method proceeds, e.g. $f(x_{100}) < f(x_1)$. Many optimization methods have been proposed and their effectiveness has been demonstrated in a variety of contexts. Some optimization problems can be solved analytically using derivatives or other means. Other problems can be effectively solved by iteratively following the gradient of the objective using methods such as Newton-Raphson [163] or conjugate-gradient descent [90]. When derivatives are not available, they can sometimes be estimated. On many practical problems, particularly those that involve complex simulations, precise estimates of the gradient are highly variable or unpredictable, or they might be too expensive to obtain. In addition, derivative-based methods are *local optimizers* that find a *local optimum* rather than the true *global optimum*; that is, derivatives can only be used to optimize in the neighborhood of a starting point. If an objective function is particularly bumpy or *multimodal*, as in Figure 1.1, then a derivative-based method must be restarted many times with different starting points, or a derivative-free method may be attempted.

Derivative-free methods use trial and error to locate the optimum. Direct search methods test every possible direction from current best solution and then move iteratively in the general direction of the unknown gradient [111].



(a) unimodal (b) multimodal, periodic (c) multimodal, aperiodic

Figure 1.1: Three example objective functions. Figure 1.1(a) is *unimodal* and possesses a unique minimum, easily located analytically or by gradient methods. Figure 1.1(b) is *multimodal* but *periodic*. Gradient methods will fail, but the periodicity can be used to locate the optimum. Figure 1.1(c) is multimodal and irregularly structured. Such problems can be difficult to solve, particularly in high dimension.

Genetic algorithms mimic Darwinian evolution by maintaining a population of solutions that are combined and varied iteratively in a manner that prefers to keep solutions with a higher score on the fitness function [77]. Monte Carlo methods such as simulated annealing sample a special Markov chain that theoretically converges on the global optimum, though with many practical caveats [109]. More recent evolutionary algorithms are based on various natural analogies, from the flocking of geese [62] to the foraging of ants [58] or the functioning of the human immune system [63]. A more thorough review of existing optimization methods is provided in Chapter 2, but this brief summary gives the reader a sense of the overwhelming number of different approaches to the optimization problem. The challenge taken on in this dissertation is to organize all these methods with an analytic approach.

1.3 Spaces of Optimizers

In the face of this variety of optimization methods, one may wonder whether a framework that unifies them will be so abstract that it has little practical meaning. A cursory study of these methods gives the first impression that the set of all optimizers for a particular search space is fundamentally discrete and unstructured, and that there is no apparent relationship between any two arbitrary optimizers. This dissertation aims to dispel this impression by presenting a mathematical analysis that reveals to the contrary that the set of optimizers for a fixed search space is highly structured in mathematical terms. It is, in fact, a closed, convex set within a normed vector space with well-formed notions of distance, continuity, and limits.

In a finite search space with finitely many output values, the structure of the space is easy enough to understand. A search space is finite if there are only finitely many configurations or inputs to the system. Each of these inputs may be assigned a natural number, so that the inputs are number from 1 to N for some $N < \infty$. The optimization task can be solved by testing each of these inputs in turn; once all inputs have been tested, the optimal input must be known.

An optimizer selects the order in which each possible input is tested. The optimizer may determine the order for later inputs based on the output values for earlier inputs. For instance, if the optimizer has proposed input number 3 as the first input to test, then it may choose to examine input number 4 as the second input to test if the objective value for 3 was negative but might instead choose input number 5 if the objective value for 3 was positive. The optimizer may randomize its choices, but for this example, assume that an optimizer makes only deterministic choices. Additionally, assume that the optimizer does not repeat itself. Since there are only finitely many inputs and outputs, there are only finitely many ways in which an optimizer may order the inputs for testing. Therefore, there are only finitely many optimizers on this space.

Simplifying even further, consider the subset of optimizers that do not depend on the output values at all. These optimizers merely specify at the outset an order in which the inputs will be tested. Suppose that there are only 10 system configurations. Then there are exactly $3,628,800$ such optimizers (i.e. deterministic optimizers that do not vary with output and do not repeat points). Each of these optimizers can be represented as a sequence of 10 numbers, such as:

$$1, 2, 5, 7, 8, 9, 6, 3, 4, 10$$
$$1, 2, 7, 5, 8, 9, 6, 3, 4, 10$$
$$5, 7, 1, 3, 4, 9, 8, 10, 2, 6$$

It is plain to see that the first and second of these optimizers are more similar to each other than to the third optimizer. In fact, one can define a distance metric between any two optimizers in this subset by counting the minimal number of entries that must be swapped in order to convert one optimizer into another. The distance between the first and second optimizer above under this metric is one. The distance between the first and third optimizers is seven. The maximum distance between any two of these optimizers is nine. Far from having no structure at all, the set of output-independent, non-repeating optimizers on a finite space is at least a metric space.

Suppose that an optimizer is allowed to depend on the outcome of objective evaluations, but is still deterministic. Then an optimizer may be specified as a function that takes as input a finite sequence of pairs containing the input value and the corresponding objective evaluation and produces as output the identity of the next input to test. Since the outputs were specified to be finite, they may be numbered as well. There are only finitely many sequences of such pairs no longer than N, and thus an optimizer is defined by a table of input-output associations. For example, if there are M outputs, numbered 1 to M, then an individual optimizer might look like the following:

evaluation history	next choice
$\emptyset$	1
(1,1)	3
(1,2)	6
...	...
(1,M)	7
(1,1),(3,1)	6
(1,1),(3,2)	2
...	...
(1,M),(7,1)	2
...	...
(1,1),(3,2),(2,6),(5,4),...,(N,5)	1
...	...

Notice that not all sequences need to be considered for a deterministic optimizer, but only those sequences that the optimizer will produce on some objective function. So only those sequences that begin with input 1 are valid for the optimizer above, since optimizer always tests input 1 first when presented with the empty sequence, $\emptyset$. This formalization is essentially identical to the one used by Wolpert and Macready in their paper introducing the No Free Lunch theorems for optimization [218].

The number of entries in such tables is bounded above by $T = N!NM^N$ since the inputs cannot repeat but the outputs can. The extra factor of N reflects the fact that input sequences can have length 1 to N; a tighter bound is possible but is unnecessary here. Thus even if optimizers are allowed to consider objective outcomes, the number of deterministic optimizers is finite and bounded above by T^N, allowing each entry to take on all N possible outputs.

Most importantly, all non-repeating deterministic optimizers on any finite search space can be represented in this table format, regardless of the rationale that led to its formulation. It does not matter whether the algorithm is described by a search heuristic or a biological analogy. Once it is reduced to a table like the one above, it is just another optimizer in the space. Furthermore, one can characterize the distance between any two optimizers as the number of edits that must be made to the table for the first optimizer to convert it into the second. It is also reasonable to speculate that two optimizers with similar tables will perform similarly on the same objective. Further, if the search domain has a known topology, then optimizers may be compared even more meaningfully by incorporating the topological structure over the outputs into the distance metric over optimizers.

The analysis in this section provides an example of the kind of structure that may be observed in optimizer spaces. Fundamentally, an optimizer is a means of selecting which input points will be evaluated next given the inputs evaluated so far and the outputs that resulted. By analyzing the outcome of these selection mechanisms independent of the descriptions and procedures used to obtain them, it is possible to compare any two optimizers on practically any search domain and objective.

It should be evident that even with substantial restrictions, such as a finite search space and deterministic, non-repeating optimizers, a general formalization of optimization methods is an ambitious project. Including randomized optimizers is not difficult. Every run of a stochastic optimizer produces a single input-output sequence, and thus a stochastic optimizer may be regarded as a distribution over the deterministic tables described above, that is, as a (very long) probability vector. However, allowing repetition or infinite search domains requires more powerful mathematical tools. Non-repeating optimizers may produce infinite sequences of inputs without observing the objective value of all inputs, and thus the tables above may require infinitely many entries to represent them. And infinite spaces can certainly not be studied by reasoning about lists and tables.

It is important to justify for why infinite spaces deserve to be studied at all. One might argue that only finite representations are computable, and so the table representations above should suffice for formal analysis. While it is true that digital computers can only represent and manipulate finite objects, many optimizers are designed to search mathematical spaces that are formally infinite. It does not make sense to limit the formal analysis of these optimizers to their finite, computable encodings. Ultimately there are two reasons to study infinite spaces directly. The first reason is that by considering the native topology of the problem, one avoids distortions that may be introduced by projecting the topology into a finite approximation. Secondly, an analysis that accounts for infinite spaces is in some ways simpler than a finite analysis because of the availability of analytic tools developed by the mathematical disciplines such as topology, measure theory, and abstract algebra.

A measure-theoretic analysis of optimization methods on topological spaces is undertaken in Chapter 3, presenting the first abstract treatment of optimization that allows for the simultaneous analysis of general optimizer spaces independent of procedural descriptions and without substantial simplifying assumptions (e.g. finiteness, lack of repetition, etc.). It applies the concepts above to infinite spaces, defines the terms that will be used, and lays the groundwork for subsequent analysis. Chapter 4 applies this framework to population-based optimizers, and Chapter 5 studies their continuity properties. What will be accomplished with this formalization is discussed next.

1.4 Optimizer Performance and No Free Lunch

Ordinarily, one is not interested in the similarity of optimizers in terms of how they select inputs. Rather, one seeks an optimization procedure that prioritizes input points with high quality. Whenever an optimizer proposes an input configuration to evaluate, the optimizer makes an error whose magnitude is given by the difference between the optimal output value and the output value for the proposed input. A good optimizer performs well on a problem if it minimizes its errors very quickly. A performance criterion specifies what kinds of errors are salient and on what time scale the errors are to be considered. Formal definitions of performance criteria are given in Chapter 7. These definitions are accompanied by experimental results for a variety of the optimization methods introduced in Chapters 2 and 4.

The obvious next question is whether there is some optimizer that outperforms all the others, not just experimentally, but theoretically. An initial answer is given by Wolpert and Macready in the well-known No Free Lunch theorems for optimization [218]. In finite search spaces with finitely many outputs, all optimizers have the same average performance over all possible objective functions. That is, no optimizer is better than any other; good performance by an optimizer on one objective is paid for with bad performance by the same optimizer on a different objective. Even an optimizer constructed to perform poorly will perform well on some objectives.

This theoretical result comes with a major qualification. The result was only proven for the case when each possible objective function is equally likely to occur. Suppose someone has devised a novel optimization method. In order for No Free Lunch to hold under Wolpert and Macready's proof, one must assume that the novel method is going to be tested against some arbitrary, unknown objective, selected according to a random procedure. If some objective functions are more likely than others under this procedure, then an optimizer that does better on the more common objectives may outperform optimizers that perform well on the less common objectives. Wolpert and Macready conjectured that No Free Lunch would hold for many if not most other random procedures for choosing a test objective. In fact, as will be shown in this dissertation, the opposite is true. No Free Lunch actually holds for very few such random procedures, and those in which it does hold are philosophically unsavory, as will be discussed below.

In Chapter 9, this dissertation expands No Free Lunch from finite spaces to arbitrary measure spaces, which can be uncountably infinite. In doing so, one encounters a substantial problem: There is no obvious way to average over all objective functions on an infinite space without preferring some functions over others. The resolution to this issue is the key achievement of this dissertation. In short, it turns out that the conditions in which No Free Lunch theorems hold can be explicitly stated for both finite and infinite spaces. No Free Lunch only applies when the random procedure for selecting test objectives satisfies two properties. First, the objective values under the test procedure must be identically distributed at each input point. Second, and more importantly, the output values obtained from evaluating the test objective at any particular sequence of input points must have no value for predicting the output value of the test objective at any other point. This property will be termed *path independence*. A random procedure for selecting test objectives produces a No Free Lunch result if and only if the procedure is identically distributed and path independent.

Consider what it means for a random optimization problem to be path independent. When optimizing such an objective, prior evaluations are useless for guessing the outcome of future evaluations. Thus the order in which input points are evaluated is irrelevant. It is impossible for an optimizer to learn anything about such a problem, because the problem reveals nothing about itself. In order to accomplish this feat, the random procedure must scramble the relationship between inputs and outputs to the point that the relationship is fundamentally incompressible. Thus there can be no rule to represent the selected test objective that is smaller than an enumeration of all input-output pairs. The world of No Free Lunch is preeminently unstructured and unlearnable; it is the fuzz between the channels on an old television set.

In small, finite search spaces, the assumption of path independence may make sense. There is no obvious way to compare categorical values with each other, and so one may as well presume that they are arbitrarily interchangeable. However, as soon as the inputs or outputs take on some kind of local structure, the assumption of path independence falls apart. In real problems, concepts such as locality, periodicity, and regularity are important. As soon as such conditions hold, No Free Lunch fails. A simple example of a random test procedure that violates No Free Lunch is the standard Brownian Motion, commonly used to model physical processes involving the diffusion of particles within a substrate, such as the expansion of coffee grounds in water. The position of a single particle at each time step forms an objective that has unbounded variation but is locally predictable with high probability. Another example of such a random test procedure is the Solomonoff's universal prior, which prefers functions that are easily computable over functions that are difficult to compute [190]. A random test procedure can be quite general without being subject to No Free Lunch.

This point of view challenges the philosophical idea that the world is unknowable at its core. If one views the universe as a random test procedure generating a variety of test objectives, then one does not expect to encounter problems in which the outcome in one situation is utterly unpredictable on the basis of previous outcomes in similar situations. This expectation of regularity is not merely utilitarian. It is not sufficient to object that humans expect the world to behave predictably because they have no choice but to do so. To make such an objection is to suggest that every decision made by humans that succeeds is purely serendipitous. While it is true that many of the more complex aspects of human life are subject to severe variability and unpredictability, it is nonetheless the case that many of the everyday aspects of life are highly predictable. When a man takes a step, his foot does not fall through the ground in front of him as it does through the air. The sun proceeds regularly through the sky, and when it sets at night, the time of its rising may be predicted precisely. Apple trees do not produce peaches, and a peach seed will not grow into an apple tree. In these and in many other situations, prior experience is a strong predictor of future outcomes. The very experience of humans as learning machines disproves the claim that the world is unlearnable.

Thus even as this thesis reaffirms No Free Lunch in a wider sphere, it rejects the popular interpretation of No Free Lunch that claims that any particular optimizer is just as good as any other. This claim is categorically false in the context of real-world problems. It is also false on computable problems on infinite spaces, not just real-world problems, since such problems must have finite representations to be computable. The existence of a finite representation is a form of regularity that invalidates No Free Lunch. In sum, some optimizers are better than others when averaged over all possible problems of interest. General-purpose learners exist.

Just because some optimizers are better than others does not mean that one particular optimizer performs best on all common objectives. As the random procedure for generating test objectives changes, the optimizer with the best average performance changes as well. The space of optimizers and the space of random test procedures are in duality. Even if an optimal optimizer for a particular random test procedure can be found, there is no way to know that the chosen test procedure accurately reflects the likelihood of actual problems of interest. In general, specific solutions to specific problems will almost always perform better than general solutions that work on many problems. This fact has often been raised as an objection to the search for general-purpose optimizers. This objection ignores the effort that human researchers put into finding such specific solutions. In practice, specific solutions are usually identified as the result of a general problem-solving methodology that relies on human learning capabilities. Although specific problem-solving strategies are to be preferred for specific problem classes, general-purpose learners are still necessary to discover such strategies.

1.5 Information-Maximizing Optimization

One may regard the optimization process as a zero-sum game in the sense of von Neumann's Game Theory [205]. One player selects an optimizer, and his adversary selects a random procedure for generating objectives. As is proven in Chapter 10, this game is biased in favor of the second player, who can always choose a selection procedure subject to No Free Lunch, since such a selection procedure always exists. If the strategy of the second player is fixed, however, then the first player must select the best optimizer for a fixed strategy. The best choice the first player can make is to play a strategy that minimizes its error. One way to minimize error is to utilize the conditional expectation of the objective function given the outcome of previous objective evaluations. The conditional expectation estimates the true objective function with minimal variability, and variability is directly correlated with optimizer errors. The conditional expectation may or may not be computable in any particular case. If it is not computable, then an approximation may be used. This sort of approach can be viewed as an information-maximizing approach, where points are to be selected in a manner that minimizes the variability of optimizer errors. There is reason to speculate that the optimal optimizer pursues a strategy that maximizes its use of available information and structures its search to improve its access to useful information.

Following this line of thought, this dissertation proposes a new method named *evolutionary annealing* in Chapter 11. Evolutionary annealing is an efficiently computable method for generating strategies that are roughly based on the conditional expectation of the objective function given prior evaluations. Like simulated annealing, evolutionary annealing approximates samples from an increasingly sharp Boltzmann distribution, asymptotically focusing on the global optima. Procedurally, evolutionary annealing resembles an evolutionary algorithm, since it proceeds in phases of selection and variation. Evolutionary annealing selects previously observed points probabilistically in proportion to their fitness in a way that asymptotically samples from the Boltzmann distribution. Then, the selected point is randomly altered to produce a new evaluation point. Evolutionary annealing is provably convergent to the global optimum under certain conditions. The proof is based on a martingale analysis that shows that the global optima become increasingly likely as the information about the objective functions is refined. Experimentally, evolutionary annealing compares favorably with other common optimization methods in a Euclidean search space (Chapter 12), based on the performance criteria presented in Chapter 7.

Evolutionary annealing is a template for new optimizers, and can be applied to arbitrary measure spaces; one needs only to specify the mechanism for generating new points from previously observed ones (in evolutionary terms, the mutation process). To demonstrate this versatility, evolutionary annealing is used to develop a novel procedure for learning artificial neural networks in Chapter 13. Artificial neural networks are parameterized functions representing a network of artificial neurons [88]. The artificial neurons are connected to each other by artificial synapses that are represented by a single real number termed the *weight* of the connection. An artificial neuron computes a function by taking a weighted sum of its input values and passing the sum through a nonlinear squashing function. The network as a whole computes a function by treating the output of a subset of neurons as the output of the function. Arbitrarily complex functions can be represented by wiring sufficiently many neurons together in different ways [49, 185]. Because a neural network computes a function, the space of neural networks can be used as a proxy to search for dynamic control functions, such as those needed to solve the driving task presented in Section 1.2 above.

In Chapter 13, neuroannealing applies evolutionary annealing to the task of learning a neural network to solve control problems. Neuroannealing compares favorably with other methods for training neural networks, and solves some tasks that require complex networks more effectively than previous methods.

In the final analysis, however, choosing a good optimization method requires an understanding of the particular optimization problem to be solved. Evolutionary annealing is an interesting new optimization method based on thorough use of available information. But it is still not the best choice for every optimization problem. Thus the broader contribution of this dissertation is to provide tools that can be used to assess which methods are the proper methods to use for a particular problem, intuitively, theoretically, and experimentally.

1.6 Guide to the Reader

A general mathematical analysis of stochastic optimization methods as undertaken in this dissertation requires mathematical tools that may be unfamiliar within the artificial intelligence and machine learning communities. There is not sufficient space in this dissertation to provide the necessary mathematical background for the formulae and proofs that follow. In particular, basic familiarity with limits, probabilities, set theory, and real analysis is assumed. Most importantly, the formalization of optimizers that is undertaken here is built on top of measure theory [43, 83] and functional analysis [8, 23], with some elements of topology [144], as well as probability theory, martingales, and stochastic processes [30, 38, 105, 126].

In order to prevent this text from becoming a sequence of impenetrable formulae and abstruse theoretical pontification, the definitions, theorems, propositions, and proofs have been infused with connective narrative that should clarify the intent and significance of the more mathematical portions of the text. To a large degree, it should be possible to obtain a workable sense of what has been presented by reading the narrative sections while skipping the proofs and much of the mathematical detail.

Although the author has made sincere efforts to present the material that follows with a proper level of theoretical depth and rigor, he readily admits that his background and experience primarily reside in practical aspects of computation. In addition, however, while the basic readership is assumed to be mathematically literate, the audience of this dissertation is, like the author, assumed to have a primary interest in computation. Although the mathematical concepts in this text are drawn from disciplines of advanced mathematics, the goal is to produce results that are of practical interest and benefit, and analytic excursions have been limited to those topics that are necessary for proving these results. In particular, integrability, measurability, and even finiteness are often blithely assumed. There is little if any mention of standard analytic topics such as compactness or dense sets, nor is there substantial discussion of convergence over sequences of optimizers or methods of approximating optimizers.

Those familiar with the subject matter who wish to skip directly the most significant contributions of this dissertation should read the following sections. The formal setting adopted for this research are given in Section 3.2 and the description of the normed vector space containing all optimizers is contained in Section 3.4. Particular attention should be given to Section 3.2.3, where the notational convention used throughout the text are defined. Performance criteria are defined at the beginning of Chapter 7, and Chapter 8 describes the experimental performance of a variety of popular optimization methods. The formal proofs of No Free Lunch and the characterization of function priors subject to No Free Lunch are found Chapter 9. The implications of these theorems are expounded in Chapter 10, which also introduces the information-maximization principle as a means of identifying the optimal optimizer for a particular function prior. The basic evolutionary annealing algorithm is presented in Chapter 11, and Chapter 12 presents an experimental analysis of its performance in Euclidean space. These chapters form the core material of the dissertation.

The other chapters contain material that, while significant, may be of less interest to particular readers. Chapter 4 discusses how particular evolutionary algorithms fit into the formal framework. Chapter 5 provides tools to aid in determining when optimizers are continuous, demonstrated by proving the conditions for continuity in existing optimization methods. This material is crucial to the proof in Chapter 7 that optimizer performance is continuous as the objective function changes, but is not otherwise used later in the text. Chapter 6 discusses the relationship between the sequence of points generated by an optimizer and the decisions made by the optimizer at each time step. This chapter also reviews aspects of stochastic processes that are needed for the proofs of No Free Lunch in Chapter 9 and formulates certain equations that are referenced repeatedly in later chapters. Chapter 13 shows how evolutionary annealing can be applied to train neural networks.

With these guidelines in mind, the reader will hopefully discover in this dissertation a new way of thinking about optimization methods that has the potential to bridge the divide between advocates of different optimization methods and to enable a proper assessment of the value of each method.

1.7 Conclusion

Let us conclude with a brief review of the salient points of this dissertation. One might think of the space of optimizers as a large, unexplored territory with pockets of civilization representing well-known and deeply studied forms of optimization, such as gradient, Monte Carlo, or evolutionary methods. However, the space of optimizers is at least as large as the space of objective functions, and many of the unknown and unstudied optimizers may prove to have practical uses. Although the No Free Lunch theorems place some bounds on the degree to which different optimizers can be compared with each other, these bounds are weaker than has been supposed. It is thus possible to develop a rich theory of optimizer performance.

The currency of this unexplored land is information – prior information about the function being optimized and information obtained from evaluating the objective. The best computable optimizer for an unknown objective is one that fully utilizes all sources of information to exclude incorrect objectives. Function evaluations provide a source of increasing information, evoking the idea of a martingale, a stochastic process of constant mean with a resolution that increases with the available information. This dissertation proposes evolutionary annealing, a martingale-driven stochastic optimizer, as an example of such a method. Evolutionary annealing is established theoretically and demonstrated to work well on several optimization problems in real vectors and neural networks. More importantly, evolutionary annealing is a new type of optimization method that is typologically different from existing optimization methods, but that arises organically from a functional analysis of the space of optimizers.

As the number of optimization methods proliferates, it is increasingly important to provide a theoretical structure within which these methods can be organized and meaningfully compared. The functional analysis employed in this dissertation is a step in this direction, and it is anticipated that analyses of this type will become increasingly important and useful in the years to come.

This dissertation is a modest attempt at the following three goals: (1) to provide a general framework and terminology for analyzing the class of iterative stochastic optimization algorithms; (2) to propose a set of analytic tools and methods for comparing optimizer performance and for selecting a particular algorithm for a particular task; and (3) to demonstrate the applicability of the analytic framework by proposing evolutionary annealing as an interesting new optimization method made possible through this analytic lens. The discussion will remain primarily at the theoretical level throughout, although experimental results will be provided to demonstrate performance criteria and to establish the efficacy of evolutionary annealing and neuroannealing. These experiments notwithstanding, the focus will be on the elegance with which a wide range of optimization methods can be compared and on the surprising relationships that exist between them.

With this summary in mind, after a historical interlude in Chapter 2, the following chapters develop the basic theory of population-based stochastic optimization. It is hoped that the definitions and formalisms herein will aid the reader in identifying the similarities and differences between the wide variety of optimization methods that now exist. It is further expected that the constructs that follow will be useful for directing future research in new and profitable directions.

Chapter 2

Review of Optimization Methods

Modern optimization methods of optimization originated in the seventeenth century with the discovery of the calculus. Until the advent of the digital computer, however, analytic solutions and fast-converging iterative methods were the only practical means of performing optimization. The introduction and proliferation of computing technologies widened both the scope and the number of optimization problems. Nearly all of the optimization methods that will be described in this chapter were developed after 1950, when large research laboratories first acquired programmable computers. The vast majority of common optimization methods were developed after the advent of the personal computer around 1980. Thus a history of optimization methods is necessarily a history of computational methods in optimization, since most of the problems and solutions described in this chapter could not be seriously posed or tested without the computing machinery that is widely available today.

2.1 Overview

This chapter presents a brief survey of the primary areas of research in optimization. The best-known methods are based on following the derivatives of an objective. Such gradient methods usually converge quickly and accurately on unimodal objectives. However, these methods may not be appropriate for objective functions where the gradient must be estimated from noisy data, or where the objective function is particularly rough, causing gradient methods to converge to a suboptimal point.

Direct search methods work on some problems where gradient-following methods fail. These methods exhaustively search all possible directions on an increasingly refined grid over the search space. They ultimately follow the gradient without estimating it, but at an exponential cost in speed.

Gradient-based methods have a major drawback: They tend to find only local optima. If one wishes to find global optima, one may run gradient methods multiple times on random points. But then the question arises of how random points should be chosen. Simulated annealing avoids gradients entirely by embedding the optimization problem inside of a sequence of probability distributions that asymptotically favor the optima. Simulated annealing still explores the search space one point at a time, but unlike gradient methods and direct search, the point is allowed to explore regions with lower objective values, permitting this method to cross hills and valleys in the objective function in search of the true global optimum.

Many of the previous descriptions have implicitly assumed that the space being searched is a continuous space. A large number of problems are in fact discrete, and many of these can be represented as a search over a graph structure. This field is known as *combinatorial optimization*. Many important optimization methods are applied specifically to these problems, such as greedy hill-climbing, simulated annealing, and genetic algorithms.

Evolutionary algorithms represent a different approach to optimization that seeks inspiration from biological processes and analogies. Darwinian evolution was the primary motivation for early research in this field; later work branched into a variety of biological analogies under the moniker *natural computation*. In the past two decades, rigorous mathematical explanations of the core evolutionary algorithms have been developed that make it possible to assess of the capabilities and limits of evolutionary methods. These analyses have also pushed the evolutionary computation community into two distinct camps: those who focus on simulated and artificial biology as a testbed for computational innovation and creativity, and those primarily interested in static function optimization. The research of the latter group has produced mathematically explicit, quasi-evolutionary methods that quintessentially represent a transition towards more mathematical representations: estimation of distribution algorithms, natural evolution strategies, and differential evolution. The present dissertation continues in this vein, with a goal of unifying evolutionary computation with static optimization, while recognizing that the study of computational creativity and the optimization of dynamic functions is a separate topic that is interesting in its own right.

Explicitly excluded from this review are componential search domains such as boolean clause satisfaction (SAT), where the structure of the domain favors approaches that break the problem down into parts that can be independently analyzed.

Instead, the methods that are described here are chosen to represent what has been termed *black-box optimization*, in which little substructure is available to the optimizer up front. On closer inspection, one finds that this distinction is somewhat contrived, since information about the objective can be embedded into most successful black-box methods. In addition, the domain-specific algorithms can be subsumed in to the formalisms of the following chapters; the formal approach does not in any way prevent it. However, to save space and to promote clarity, this text will focus on the tradition of black-box methods.

2.2 Gradient Methods

Gradient-based optimization methods have a long history and continue to be widely used due to their fast convergence to accurate local optima. This section reviews the origins of gradient-based optimization, leading up to a discussion of its modern variants.

2.2.1 Early Methods

Early optimization methods focused on real-valued functions of one or more real variables. The first published work addressing this problem is Pierre de Fermat's *Methodus ad disquirendam maximam et minima*, written in 1638, in which Fermat presented a method for locating the minima and maxima of a function that corresponds to what is now termed the *first derivative test* [55]. Given a real function over an interval of the real line, the extrema must either lie at the boundary, at the non-differentiable points, or at the points where the function's derivative is zero, i.e. $f'(x) = 0$. If these points are few in number, then the function can be evaluated at each of them to identify the true maximum or minimum. The *second derivative test* provides a means of determining whether internal points are minimal or maximal. Using these tests, the global extrema of a function can be determined analytically for many important objectives. However, the method assumes that the non-differential points are identifiable, and it requires that the objective function be stated in closed form as an equation. Most importantly, one must be able to locate the zeros of the derivative, a difficult task even for many equations easily stated in closed form.

Fermat's method can also be extended to functions of several variables, but the restriction to intervals limits its applicability. The introduction of Lagrangian multipliers in the eighteenth century provided a means for enforcing more complex constraints [121]. The modernized refinement of Lagrange's method, the Karush-Kuhn-Tucker conditions, remains an important technique in constrained optimization [106, 119].

2.2.2 Newton's Method

By 1669, Isaac Newton had discovered an iterative method for locating the zeros of a real function, now known as *Newton's method* or the *Newton-Raphson method*. Given a continuous real function $f(x)$ and a starting point x_0, the sequence (x_n) defined recursively by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

converges to a root of f, i.e. $f(\lim x_n) = 0$. Building on the derivative test methods of Fermat, this result implies that the sequence

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} \tag{2.1}$$

converges to an extremal point of f [208]. Equation 2.1 is not actually due to Newton or Raphson. Newton devised an equivalent derivation as a sequence of polynomials, and it was Raphson who presented the scheme as an iterative search along the real line [163]. Surprisingly, neither Newton nor Raphson recognized the relationship of the method to the calculus, but instead used the method only on polynomials; the generalization to arbitrary functions in Equation 2.1 was not recognized until later in the eighteenth century.

When it applies, the Newton-Raphson method converges quickly. The rate of convergence is quadratic, that is, the distance from the iterate x_n to the local optimum x is inversely proportional to n^2. Unfortunately, pure Newton-Raphson has limited applicability. The objective function must have a continuous first derivative and a finite, nonzero second derivative. The starting point x_0 must be sufficiently close to the extremum x, and if there are multiple local optima close together, then convergence will be slower. In some cases, the iterates may enter a limit cycle. Finally, only a local optimum close to the starting point x_0 will be found, and there is no way to choose the starting point x_0 without analyzing the objective function or sampling from it. If the derivatives are not available in closed form, they may be estimated by sampling points near x_n and applying the *finite difference* method. The approximation of the derivative with finite differences is termed the *secant method*; a version of the secant method, the Rule of the Double False Position, dates back to ancient India and China [151]. The secant method has a linear convergence rate rather than quadratic.

Newton-Raphson can be generalized to real functions of more than one variable. In this case, the gradient $\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$ and the Hessian matrix $\nabla^2 f = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]$ must be computed. Then the iteration is given by

$$x_{n+1} = x_n - \eta_n \left[\nabla^2 f(x_n)\right]^{-1} \nabla f(x_n) \tag{2.2}$$

where $\eta_n > 0$ has been introduced as a time-varying *step size* or *learning rate* to aid convergence as described in the discussion of line search below. In the multi-dimensional version, the gradient functions must each be Lipschitz continuous, and the Hessian must be invertible. If the conditions are satisfied, there is a neighborhood of each local optimum such that the generalized Newton's method converges to that local optimum for all starting points contained within the neighborhood.

Newton's method is one of the most effective known optimization methods, but it applies very narrowly due to the strict conditions of continuous differentiability and an invertible Hessian. Various approximations relax some of these requirements and replace the Hessian and/or the gradient with approximations and simplifications. These approximations were primarily developed during the mid-twentieth century and constitute the most popular and widely known optimization methods today. Of these, the most significant are *gradient descent* and *line search*.

2.2.3 Gradient Descent

Equation 2.2 is difficult to compute because of the Hessian. However, an effective method can be developed by omitting the Hessian. Given a point x_n, the Taylor expansion of f around x_n is given loosely by

$$f(x_n + \eta_n) = f(x_n) + \eta_n \nabla f(x_n) + o(\eta_n)$$

where, as usual, $o(\eta_n)$ indicates a term that is asymptotically obliterated by η_n (i.e. $o(\eta_n)/\eta_n \to 0$). If the step size η_n is small, then $\eta_n \nabla f(x)$ is much larger than $o(\eta_n) \approx \eta_n^2 ||\nabla^2 f(x_n)||^2$. In this case the final term can be ignored with minimal error, and the iterates can be computed by

$$x_{n+1} = x_n - \eta_n \nabla f(x),$$

where the step size becomes smaller as n increases. The sequence x_n defined by this iteration typically converges to a local minimum of f; a local maximum can be found by reversing the sign. Because this method follows only the gradient and ignores the curvature of the objective, it is known as *gradient descent* or *steepest descent*. As a result of ignoring second-order information, the step size η_n must be small in order for gradient descent to succeed, and its convergence rate is consequently slower. Nevertheless, gradient descent is typically easy to implement and avoids the calculation of the Hessian.

Gradient descent has three common failure modes. First, the step size can be too large, causing divergent oscillation away from the optimum. This error can be avoided by choosing the step size to respect the Armijo (or Armijo-Goldstein) condition,

$$f(x_n + \eta_n \nabla x) \leq f(x_n) + c_1 \eta_n ||\nabla f(x_n)||^2,$$

where $||z||^2 = \sum_i z_i^2$ is the square of the standard Euclidean norm [9]. If the step size decreases too quickly, then the iterates can converge before reaching a local minimum. This situation can be prevented by observing the Wolfe condition,

$$||\nabla f(x_n + \eta_n \nabla x)||^2 \geq c_2 ||\nabla f(x_n)||^2,$$

with $0 < c_1 < c_2 < 1$ [216].

As a different type of failure, gradient descent may converge to a shallow local optimum rather than a deeper one nearby. A common approach to avoid this problem is to introduce a momentum factor μ and set

$$x_{n+1} = x_n - \eta_n \nabla f(x) + \mu \Delta_n$$

where $\Delta_n \equiv x_n - x_{n-1}$ [158]. The value $\mu = 0.9$ is standard. This method is often justified by the analogy of rolling a ball over the function surface. The momentum of the ball allows it to roll over small depressions, so that it only gets trapped in a large hole. In certain situations, however, the momentum factor may backfire, causing the iterates to pass over the attraction basin of the global minimum and into a local minimum.

2.2.4 Line Search

Rather than eliminate the Hessian entirely from Equation 2.2, one can replace it with a more manageable matrix. This generalization of Newton's method is known as *line search*, and it is defined by the equation

$$x_{n+1} = x_n - \eta_n B_n^{-1} \nabla f(x_n)$$

where B_n is a positive definite matrix. In this case, the quantity

$$d_n = -B_n^{-1} \nabla f(x_n)$$

is termed a *descent direction*. The sequence (x_n) will still converge to a local optimum of x provided that the inequality

$$-\nabla f(x_n)^T d_n > 0$$

holds. This inequality guarantees that x_{n+1} moves towards the optimum in general (i.e., the descent direction is not orthogonal to the gradient and does not move against it). Generalized versions of the Armijo and Wolfe conditions can be used to select the step size [111]. Importantly, the step sizes do not necessarily decrease and may increase so long as $f(x_{n+1}) < f(x_n)$. A *backtracking line search* may attempt several values of η_n until this monotonicity condition is satisfied.

Line search is a generalization of both Newton's method and gradient descent. In Newton's method, the matrix B_n is given by the Hessian, $\nabla^2 f(x_n)$, whereas in gradient descent B_n is the identity matrix. The key intuition is that the matrix B_n linearly transforms the search space in a manner dependent on x_n. In the case of gradient descent, no transformation is performed. The Hessian transforms the space to optimally reflect the curvature of the objective function at the current iterate. One may imagine that between these extremes there exist transformations that still tailor the iteration to the shape of the objective function at point x_n but are simpler to compute than the Hessian. One popular method that can be characterized in this way is *conjugate gradient descent* [90]; another is to use the second derivative along only a single axis.

2.2.5 Gradient Descent in Parametric Models

The description of gradient descent in Section 2.2.3 assumes that the gradient ∇f must be computed for the objective function f. In practice, gradient methods can be applied in many situations where ∇f is not available through the use of a parametric model. For example, in the car driving task of Section 1.2, the objective function is a simulation and the search domain is a functional space. Clearly, a simulation cannot be differentiated.

Proposed solutions in this sort of task are often formulated as parameterized functions, so that a solution is of the form $c(x; \theta)$ where x is the environmental state and θ is a set of parameters drawn from a parameter space Θ. For example, in a neural network the parameters are the weights between connected artificial neurons. The objective function can be rewritten as $\tilde{f}(\theta) = f(c(\cdot; \theta))$. In many cases, the parameterized form $\nabla\tilde{f}(\theta_n)$ depends only on some statistics $(t_1^n, \dots, t_m^n)$ gathered during the evaluation of f; that is, $\nabla\tilde{f}(\theta_n) = h\left(t_1^n, \dots, t_m^n\right)$. This situation occurs commonly when the objective function is to minimize interpolation error on a given set of input-output pairs, where the statistics gathered are the interpolation errors for each input. Thus in order to compute x_n using gradient descent, one does not need $\nabla_\theta f(c(\cdot;\theta_n))$ but only $t_1^n, \ldots, t_m^n$, and the particular statistics needed depend in part on the objective function and in part on the parameterization.

Thus parameterized gradient descent and direct gradient descent differ in the type of information they extract from the objective evaluation in order to update the next solution. This distinction is important for reasons that will become more clear during the discussion of information restrictedness in Chapter 3.

2.3 Problems with Gradient Methods

The previous section gave a rough but thorough review of gradient-based methods. Since most of the text that follows focuses on gradient-free methods, the amount of material allocated to gradient methods in this chapter requires some justification. First of all, if gradient methods work at all, they work accurately and quickly. Their failure modes are also well understood and can often be identified a priori or during operation.

There are also several reasons why many gradient-free methods exist and continue to be invented. Perhaps the most obvious one is that many optimization tasks are performed on non-continuous search spaces. A prime example is the Traveling Salesman Problem (TSP), in which a salesman is tasked with finding a route that visits each city on a map exactly once using existing roads. Solutions to this problem can be represented in multiple ways, but the representations typically do not induce a continuous search space. Non-continuous tasks require searching on various spaces, including trees, permutations, graphs, subgraphs, and binary codes. Objective functions on these spaces do not have derivatives and cannot be searched with gradient-based methods.

There are also reasons why gradient-free methods can be preferable even in continuous spaces with differentiable objectives. The main one is that gradient-based methods are local optimizers. In the optimization literature, when it is said that an optimizer converges to an optimum, what is typically meant is that the optimizer produces a sequence of points that converges to some local optimum in the limit. Given an objective function f, a point x is a *local minimum* (or a *local maximum*) if there is some open neighborhood N containing x such that $f(x) \leq f(z)$ for all $z \in N$ (for a maximum, $f(x) \geq f(z)$). If this inequality is satisfied for the entire search space, then the point is a *global minimum* (or a *global maximum*). A gradient-based method will converge to a local optimum that is determined by the starting point and the step size. A local optimum may be a high-quality solution to the optimization task, or it can be of low quality. Thus a local optimum is sometimes a sufficient solution, and sometimes not.

The local optima of a function divide the search space into a disjoint set of *attraction basins*. An attraction basin of a dynamic system is the set of initial conditions for which the system converges to a particular attractor. Gradient-based optimization approximates a dynamical system given by

$$dz(t) = \nabla f(z(t))\, dt$$

with initial condition $z(0) = z_0$ and $t \in [0, \infty)$. In this system, the local optima are attractors, since the gradient is zero there. For a given local optimum x, the attraction basin of x is the set $\{z_0 \mid x = \lim_{t\to\infty} z(t)\}$. Generally, if the starting point z_0 is in the attraction basin of x under this system, then the iteration of a gradient method will converge to x unless the step size is large enough that one of the iterates steps across the entire attraction basin. Most gradient methods aim to find the local optimum of the attraction basin that contains the starting point. To study the quality of a gradient-based method on an objective, one should study the attraction basins of the objective.

Even among differentiable functions, the attraction basins of a function may have almost any shape, size, or arrangement within the search space. Figure 2.1(a) shows contours and attraction basins for the sum of ten two-dimensional Gaussian kernels with varying centers and size, given by

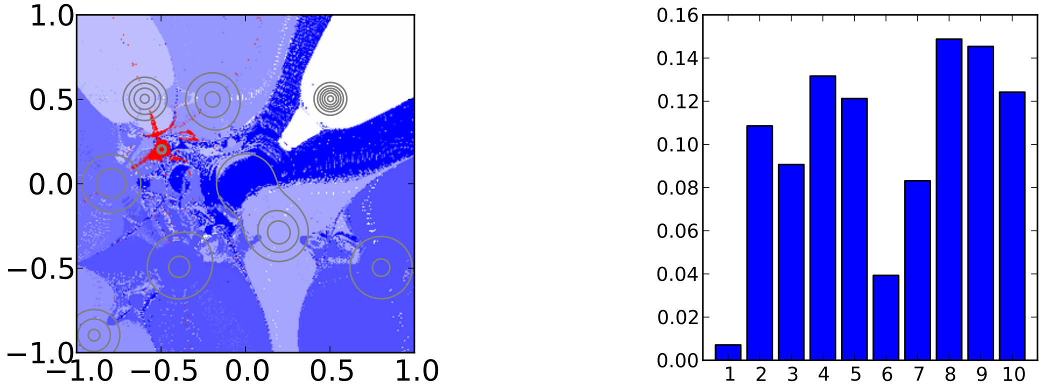
$$f(x) = \sum_{i=1}^{10} \frac{1}{2\pi\sigma_i} \exp\left(-\frac{1}{2\sigma_i^2}||x - \mu_i||^2\right).$$

This function has 10 maxima, and the basins for each were determined by running conjugate gradient descent on a 300 × 300 grid of starting points arranged from -1 to 1 on each axis. Importantly, the narrower kernels have higher objective values at the center, but correspondingly smaller attraction basins. Assume that the starting point is chosen uniformly at random with each component in the interval $[-1, 1]$. Figure 2.1(b) shows the probability of achieving each of the 10 maxima, ordered by rank with the global maximum at the left. The true optimum is located in the smallest basin and attracts only 639 out of the $90,000$ sample points. That is, if the starting point is chosen randomly, then the true optimum is found on less than one out of every 100 runs. If this experiment were repeated in five dimensions with a similar function, the chances of finding the true optimum would be much less than one in $10,000$. Therefore, the fact that a gradient-based method will converge quickly and accurately to a solution provides little comfort if the solutions achieved in this manner are severely suboptimal with high probability.

For many tasks, there is a rational means for choosing a starting point that will yield high quality. For example, if the objective is given in closed form as above, it may be analyzed. Yet for many other optimization tasks, there is no way to know at the outset which starting points will yield good results. One must choose the starting point heuristically.

In essence, the use of a gradient method converts a continuous optimization task into a discrete task. An objective with N local optima yields N possible outcomes under Newton's method. It may be that each one of these local optima are acceptable solutions, in which case the gradient search always succeeds. But if only a proper subset of the outcomes are acceptable, then the gradient search is successful only some percentage of the time. This percentage is a property of the objective and may be small.

Several methods have been proposed to improve the quality of the optima obtained by a gradient search. Each of these methods succeeds in some situations and fails in others. The use of a momentum factor was mentioned during the discussion of gradient descent. It can backfire by causing the optimizer to skip over the true optimum in certain situations. Another approach is to add random noise to the computation of the gradient. This approach is called *stochastic gradient descent*, and the random noise is usually embedded into the objective function itself as part of the task. The addition of noise can allow the gradient iteration to escape randomly from shallow local optima. The method succeeds when poor local optima are more shallow than



(a)

(b)

Figure 2.1: (a)Attraction basins for a sum of ten Gaussian kernels under conjugate gradient descent on a square region. Plots were generated by sampling a 300×300 grid to discover which local maximum results from each grid point using conjugate gradient descent. Different colors indicate regions that result in different local maxima. Contour lines are included in gray to indicate the location of the Gaussians. The global maximum is the red region in the upper left. Its area among possible starting points is small; only 639 of the $90,000$ sample points (0.7%) converge to the global maximum. (b) A bar chart showing the probability of arriving at the best local maxima, ordered by quality from left to right. Again, gradient descent is unlikely to find the best solution.

good local optima, since the variance can be tuned just enough so that the iteration escapes poor optima with high probability but is trapped by good optima. But there is no way to know ahead of time whether the poor and good optima fit this profile. An objective can be constructed in which poor optima are relatively deep and good optima are reached by a long series of shallow steps, each of which can be escaped. It is impossible to know how the variance should be set without some experimentation.

If the momentum factor and random noise fail, then another approach is to run a Newton method many times with different starting points. These methods typically require at most several dozen objective evaluations before convergence, making this approach practical. In this case, one is essentially randomizing the starting point and then sampling from a histogram like that of Figure 2.1(b). If the good optima are unlikely, then it is possible that even this method will fail.

Gradient-free methods provide alternatives that answer many of the problems brought up in this section. A full review of these methods is presented over the next few sections.

2.4 Direct Search

Direct search is a catch-all term for several gradient-free optimization methods frequently employed within the applied mathematics community over the course of the last century. According to the definition of the term, any method that relies strictly on objective evaluations in order to determine the next point to search is a direct search method [96, 111, 160, 220]. However, the term "direct search" is not used outside of applied mathematics, and it will not be applied more widely here. Some of the earliest variants of direct search were already in use at Manhattan Project in the early 1940's [111]. The most important categories in this group of approaches are the simplicial methods, represented by the Nelder-Mead algorithm, and pattern search, to which the name *generating set search* is applied following Kolda et al. [111]

2.4.1 Simplicial Methods

Simplicial optimization methods search through a d-dimensional continuous space manipulating a simplex with its $d + 1$ vertices. Because of the way the simplex moves through the search domain, this method is often referred to as the *amoeba method*. The initial simplex is built around a starting point, typically by taking the starting point as a vertex and setting the other d vertices by adding the d coordinate vectors to the starting point. The endpoints are then evaluated under the objective function. At each iteration, the simplex is be transformed in various ways.

The first simplicial method was proposed in 1962 by Spendley et al. [193]. It included two possible transformations to the simplex. The worst vertex could be reflected around the centroid of the opposite face, or the entire simplex could be shrunk towards the best vertex. Nelder and Mead introduced additional transformations [149]. In addition to being reflected, the worst vertex could be moved towards the centroid of the opposite face (contraction), or projected through it twice as far as the reflection would have accomplished (expansion). The transformation to be used is determined by a set of rules that depend on the objective values of the vertices.

The Nelder-Mead algorithm is popular because it works quickly and reasonably well. An implementation of this optimizer is included with several software packages and is widely available. However, Nelder-Mead is not guaranteed to converge, and its widespread use seems to be primarily a matter of convenience.

2.4.2 Generating Set Search

In contrast to simplicial methods, generating set searches are designed to ensure convergence to a local optimum. Generating set search maintains an accepted solution that is updated only if the objective value can be improved. The name was coined by Kolda et al, and subsumes the earlier term *pattern search* of Hooke and Jeeves [96, 111]. Recalling the definition of a descent direction in line search from Section 2.2.4, suppose that one wishes to follow the gradient in order to improve the accepted solution, but no gradient information is available. In a d-dimensional continuous space, this can be accomplished by testing the objective at $d+1$ points around the current accepted solution. If these points are chosen correctly, then at least one of them will be a descent direction for the objective.

The correct points can be generated from a *positive spanning set*. A set $B = (b_1, \ldots, b_k)$ of d-dimensional vectors is a positive spanning set if for any vector z in the space, there exists vector α with non-negative components such that $z = \sum_i \alpha_i b_i$. If $\tilde{B}$ is a basis for the space, then a positive spanning set of size $d+1$ can be generated by appending to $\tilde{B}$ a vector $\tilde{b} = -\frac{1}{d}\sum_{x \in \tilde{B}} x$. Or, a positive spanning set of size $2d$ can be generated by extending $\tilde{B}$ with the negative of each vector in $\tilde{B}$.

The simplest generating set search starts at a given point x_0 and requires a step-size parameter Δ_0. The method generates a positive spanning set B and then polls each of the directions $x_0 + \Delta_0 b$ for $b \in B$. If $f(x_0 + \Delta_0 b) < f(x_0)$ for at least one direction b, then the current solution is updated to $x_1 = x_0 + \Delta_0 b$ and $\Delta_1 = \Delta_0$. The search can either choose the best descent direction or the first discovered. The process continues until a point x_n is found such that none of the directions in B yields an improvement. In this case, $x_{n+1} = x_n$, and the step size Δ_n is reduced by some factor $\tau < 1$ so that $\Delta_{n+1} = \tau\Delta_n$. The process continues again until Δ_n falls below a tolerance, in which case the search is complete.

This algorithm converges to a local optimum because the step size Δ_n can be proved to decrease to zero asymptotically [45, 111]. It can be modified in a number of ways while remaining convergent. A search heuristic can be inserted before each polling step that evaluates any finite number of points on the grid $\{x_n + m\Delta_n b \mid b \in B, m \in \mathbb{Z}, m \neq 0\}$. When the search heuristic is successful, the step size Δ_n can be increased instead of decreased. The generating set B is allowed to depend on x_n as long as the magnitude of its vectors does not increase.

Several optimization methods that fit this template have been developed and continue to be proposed [11, 45, 202]. These methods work well in practice, but are designed to converge to a local optimum. Once convergence has occurred within a specified tolerance, the search is complete, and new local optima can only be discovered by restarting from a different point. A systematically different approach is to use a stochastic algorithm, as will be discussed next.

2.5 Stochastic Optimization

Most of the optimization methods examined so far have shared two properties. First, they have been deterministic. The resulting solution is a function of the starting point. Secondly, they converge to a single local optimum and then either terminate or become asymptotically stable. Once these methods have converged within an acceptable tolerance, they no longer explore new regions of the search space. In contrast, stochastic optimizers search the domain by randomly sampling points based on the objective value of one or more previously evaluated points. Because they move randomly, stochastic optimizer can escape local optima with some probability. As a result, they may not always converge, or they may at least explore multiple local optima prior to convergence. The most dominant method of this type is simulated annealing, which is reviewed in this section.

2.5.1 Simulated Annealing

Simulated annealing was developed by Kirkpatrick et al. in the early 1980's [26, 109]. It employs properties of statistical mechanics to locate minima of a given fitness function. The usual analogy is that of crafting a metallic artifact by repeatedly shaping it at different temperatures. At high temperatures, the metal is malleable and easy to shape, but as such the metal does not easily remain in detailed configurations. As the temperature is gradually lowered, more refined and delicate shapes become possible, but the overall shape is increasingly fixed.

At the core of the simulated annealing algorithm is the Boltzmann distribution. At time n, simulated annealing samples from a distribution given by

$$\mathcal{A}_n^f(dx) = \frac{1}{Z_n} \exp\left(-\frac{f(x)}{T_n}\right) dx, \tag{2.3}$$

where f is the fitness function, Z_n is a normalizing factor known as the *partition* *function*, and T_n is a sequence of temperatures with $T_n \to 0$. The sequence T_n is known as the *cooling schedule*. The distribution $\mathcal{A}_n^f$ will be referred to as an *annealing distribution* in this paper. Simulated annealing samples from $\mathcal{A}_n^f$ repeatedly using the Metropolis algorithm [87, 136]. The process begins with a proposed solution x. At each time step, a proposal distribution $\mathbb{Q}$ is used to sample x_n. The proposed solution x is replaced with x_n with probability $\exp\left(-\max\left\{0, f(x) - f(x_n)\right\}/T_n\right)$. For each fixed temperature T_n the algorithm will converge to a sample from $\mathcal{A}_n^f$. As $n \to \infty$, $\mathcal{A}_n^f$ converges in probability to a distribution that samples directly from the optimal points of f [109].

Subject to conditions on the cooling schedule, simulated annealing can be shown to converge asymptotically to the global optima of the fitness function [82, 223]. For combinatorial problems, Hajek [82] showed that simulated annealing converges if the cooling schedule is set according to $T_n \propto 1/\log n$. In practice, simulated annealing has been used effectively in several science and engineering problems. However, it is highly sensitive to the proposal distribution and the cooling schedule.

Whereas simulated annealing lowers the temperature T_n to zero in order to sharpen the Boltzmann distribution, *stochastic tunneling* raises the temperature to higher values in order to soften the function and lower the barriers separating the attraction basins of different local optima. Raising the temperature allows for complete exploration of the local minima of the function and may make it possible to locate the global minima. However, more thorough exploration comes at the cost of much slower convergence.

2.5.2 Justification for Stochastic Methods

Stochastic optimizers have some advantages over deterministic methods. Stochastic optimizers do not become trapped by local optima as easily as deterministic optimizers, although eventually most popular stochastic methods do converge around a single point, potentially a local optimum. However, this flexibility comes at a high price. Stochastic methods inevitably converge more slowly than deterministic gradient-based methods because they can explore in the opposite direction of the gradient. This slowdown may be exponential if the stochastic method is particularly thorough. Direct search methods suffer from the same reduction in speed, but many of them can still work faster because they provide a guarantee that a descent direction is eventually followed. Thus, before using a stochastic method, especially a non-gradient-based stochastic method, some justification is required.

Stochastic methods are valuable because they reflect the underlying uncertainty in the optimization task. As will be seen in Chapters 9 and 10, there is substantial reason to believe that the best optimizer under any fixed random test procedure is deterministic. However, if the exact nature of the test procedure is unknown, an optimizer can be made robust against the uncertainty of the test procedure by randomizing. The best deterministic optimizer on one test procedure could have poor performance on a slightly different test procedure. The primary justification for a stochastic method is the underlying uncertainty about the true nature of the problem.

If one desires to study a specific optimization task, then one might learn a great deal about the problem through exhaustive exploration. It would then always be advantageous to design an optimization method that accounts for what one has learned. However, it is important to recognize this procedure for what it is: a long, tedious, manual optimization process in which the researcher has adopted the role of the optimization method and has himself made a series of evaluations in order to remove uncertainty and increase available information. In other words, this approach reflects confidence in humans as superior optimizers. However, the skills, knowledge, and ability required by such a researcher are rare in relation to the number of optimization problems, and the "running time" of the human optimizer is often much longer than that of a general purpose optimizer. If resources and time permit, then the human optimizer is almost always preferable to automated optimization methods. But if resources and time are a constraint, then a good stochastic optimizer is a viable option. Finally, if the goal of research is to develop human-level problem solving abilities, then comparing the success of human researchers to the success of a good black-box algorithm is useful for assessing the progress of artificial intelligence.

In the end, stochastic methods are useful and can be shown to have good performance on a wide array of metrics. Deterministic or quasi-deterministic optimizers can always perform better on static problems with a sufficient amount of information. The success of an algorithm depends on its alignment with the test procedure used to evaluate it [218].

Note that the statements above do not necessarily hold for dynamic and stochastic objectives. In this dissertation, these are explicitly excluded, and the focus is on static objective functions. That is, if an objective f is evaluated at a point x, then the value $f(x)$ is fixed, i.e. subsequent evaluations must return the same value. If subsequent evaluations of $f(x)$ can change, then either the objective function is dynamic (varying over time) or stochastic. If the objective is a random function, then it seems reasonable to conjecture that appropriate stochastic methods should outperform deterministic optimizers, although this conjecture is not explored further in this dissertation.

2.6 Evolutionary Methods

Evolutionary computation is a major category of stochastic optimization method. Its origins lie in the computational simulation of evolutionary processes. The general concept of evolutionary computation has been invented independently numerous times by different researchers [15, 32, 67, 69–71, 165, 170]. This section reviews the most common elements of these methods.

2.6.1 Overview

The basic structure of a traditional evolutionary algorithm consists of a sequence of subsets of the search space, termed *populations*, with each population in the sequence called a *generation*. Population consist of *individuals*, generally represented as an array of parameters. The population for each generation is built from the prior generation through processes of competitive selection and random variation. The prior generation is ranked according to fitness, and the most fit individuals are chosen to create the next population either with minor variations called *mutations* or by combining parameters from two or more members in an operation called *crossover*. Many other variants exist, and these variants will be discussed as needed. A short historical summary of evolutionary computation follows.

The earliest research on computational simulation of evolutionary processes was published in 1954 by Barricelli in the context of a cellular automaton [22]. Friedberg and Fogel independently studied genetic programming [67, 71]. Evolution stragegies was an approach developed by Rechenberg and Schwefel to search through Euclidean space with selective competition and local variation [165, 183]. Genetic algorithms became the most dominant branch of evolutionary algorithms, having been developed independently by Bremerman [32] and by Holland and his students [15, 94, 170]. Holland championed the cause of genetic algorithms and developed a result known as the schema theorem to explain their success [95]; his work was followed by that of Goldberg, who likewise had a substantial influence in popularizing genetic algorithms [77]. Early work comparing the effects of varying algorithm hyperparameters and population transition mechanisms empirically was performed by De Jong [56]; these experiments were furthered by Brindle [33].

The schema theorem asserts that genetic algorithms would probabilistically select and refine subcomponents by making it more likely that adjacent subcomponents contributing high fitness would survive into later generations. The schema theorem was used as an argument to assert that binary representations were more efficient because binary subcomponents would be more likely to be preserved in population transitions [77, 95]. This argument assumes that problems are structured into small, uncorrelated subcomponents, when in fact real-world problems can be structured in ways that exhibit intricate long-distance dependencies [65]. In these cases, genetic algorithms are likely to struggle to find the correct problem structure. Furthermore, genetic algorithms are highly sensitive to representation in a binary setting. For example, De Jong produced an experiment comparing two different types of genetic algorithms in which a change of representation reversed their order in terms of performance [103].

Experiments like those of De Jong on the sensitivity of a genetic algorithm to its implementation details created an atmosphere of ambivalence about the prospects of tuning a genetic algorithm to solve yet more complex problems. Further, the No Free Lunch Theorems of Wolpert and Macready [218] demonstrated that, averaged over all problems, any genetic algorithm would perform equivalently to any other genetic algorithm. From that point, research turned to generalizations and abstractions of genetic algorithms, for which it was hoped that novel techniques would outperform genetic algorithms on particular problem domains where genetic algorithms fared poorly.

Many of these new algorithms were proposed as biological analogies. Collectively, they are referred to as *natural computation*. Dorigo presented Ant Colony Optimization (ACO) [58] in 1992 in analogy with the food gathering behavior of ants. For ACO, problems are transformed into a search for a path through a graph. In each generation, a population of *artifical ants* explore the graph stochastically, laying down *pheromones* that bias future searches whenever they locate interesting structure. Particle Swarm Optimization (PSO) was introduced in 1995 by Eberhart and Kennedy [62, 107] to imitate the behavior of flocks of migrating birds in choosing a direction in which to fly. In PSO, rather than randomly mutating population members, candidate solutions in the population travel through the search space with a directional momentum that is incrementally shifted towards the current best solution in the population. Artificial Immune Systems (AIS) were proposed in stages by several researches to employ an analogy with the structure of the human immune system in order to address issues such as network intrusion detection using a population of *antibodies* [25, 63]. Particle Swarm Optimization will be analyzed as an example of natural computation in Chapter 4.

In addition to the biological variants, a substantial field of research developed to learn a probability distribution governing the transition between populations in a genetic algorithm [17, 18, 86, 120]. This research culminated in algorithms that greedily learn Bayesian networks over the best members of the population; these networks are then sampled to produce the next generation [142, 155–157]. These algorithms are alternately known as Estimation of Distributions Algorithms (EDAs) or Probabilistic Model-Building Genetic Algorithms (PMBGAs) [154]. EDAs are relevant to the dissertation as an example of the application of probability to genetic algorithms and will bereviewed in further detail in Section 2.7.2.

With this historical background in mind, it will be useful to discuss the issues arising in theory and analysis of classical genetic algorithms before returning to newer, probabilistic variants.

2.6.2 Genetic Operators

Genetic operators are used to describe the substructure of population transitions in genetic algorithms. A transition from one population to the next is accomplished by applying an ordered sequence of operators to the current population. Typically, these operators include a selection operator, a crossover operator, and a mutation operator. Other genetic operators have been proposed such as diploidy, dominance, and inversion [77]. The most common genetic operators can be subsumed by a two stage process of selection and variation, where selection is a weighted choice over previously observed individuals and variation is an arbitrary stochastic perturbation of a selected individual. A formal account of selection and variation is provided in Chapter 4. The relevant background is summarized in the following paragraphs.

The role of a selection operator is to select the parents of the next generation from among the individuals whose fitness has already been measured. Selection can be treated as a probability distribution over candidate solutions observed thus far. The most common types of selection are *proportional selection* [77, 95, 154, 206], *ranking selection* [16, 206], *tournament selection* [33, 206], and *elitist selection* [56]. Proportional selection (also known as *roulette wheel selection*) is an individual-level selection operator that selects members of the previous generation in proportion to their fitness [77, 95]. Formally, for a fitness function f, individuals x, y in the prior generation, and the number of occurrences of x in the prior generation n_x,

$$\mathbb{P}^t_{\text{prop}}\left(x\right) = \frac{n_x f\left(x\right)}{\sum_y n_y f\left(y\right)}. \tag{2.4}$$

Proportional selection plays a key role in the simple genetic algorithm and in many EDAs. Note that proportional selection assumes that the fitness function is positive. Any bounded function can be shifted so as to be positive, or other monotone increasing transformations may be applied to obtain a positive function (such as exponentiation). These transformations may distort the shape of relative weight of different solutions, but they cannot alter the order in which solutions are preferred.

Ranking selection can operate either at the level of populations or individuals. At the individual level, parents for the next population can sampled randomly from the ranked population according to any monotone decreasing weighting scheme. Ranking selection differs from proportional selection in that the relative weight of population members can be altered by any method that preserves order.

Tournament selection also ranks the population, and then chooses the best individual with probability p and the k^{th} individual with probability $p\,(1-p)^{k-1}$ [33, 206].

Elitist selection guarantees that the best solution from all prior generations survives into the next population. When elitist selection is used along with a mutation scheme that eventually visits every point in the space, then asymptotic convergence is obtained with probability one [173].

Selection operators introduce competition into the population. By contrast, crossover and mutation operators provide variation to widen the scope of the search in order to discover highly fit individuals not previously in the population. A crossover operator recombines two candidate solutions to construct a third candidate solution. Crossover relies on two selection operators, one for each parent, which may or may not be the same. In addition, a crossover methodology must be supplied to determine how recombination is to occur.

The most basic crossover strategy is *one point crossover*, in which an index inside of a chromosome is selected as a split point, and each parent provides the parameters on one side of the index. The split point may be selected deterministically, or chosen according to some probability. In contrast, *uniform crossover* performs a Bernoulli trial for each parameter, and copies the parameter of the parent chosen by that trial [201].

Mutation operators are applied at the individual level to slightly vary an individual's parameters. Crossover can only alter the particular arrangement of parameter values already present in the population; mutation operators introduce new parameter values into the population. Mutation therefore facilitates detailed exploration of regions with high fitness. In binary spaces, this concept is formalized by the *mutation rate*, that is, the expected percentage of bits in the gene that will be flipped by mutation. In metric spaces, the analogous feature is characterized by the variance of the operator. If an individual can be transformed to any other individual in the space with some nonzero probability after application of the mutation operator, then an evolutionary algorithm equipped with this operator will asymptotically locate the optima of the fitness function [64, 173].

Both crossover and mutation can be construed as a family of probability distributions indexed by candidate solutions. Given an individual, crossover places nonzero probability on solutions that can result from crossing that individual with other members of the population using the crossover methodology. Mutation places nonzero probability on solutions reachable from a given individual. These intuitions are made more explicit in the discussion of Random Heuristic Search that follows.

2.6.3 Random Heuristic Search

Some theoretical issues pertaining to the simple genetic algorithm were explored by Vose [206] within a framework he introduced under the name of Random Heuristic Search. The analysis performed by Vose prefigures certain aspects of this dissertation and will therefore be reviewed in this subsection. Vose describes the simple genetic algorithm as a search in the space of probability vectors. A random heuristic search [207] consists of two repeated steps. The first step applies a deterministic *heuristic* $\mathcal{G}$ to map the current population p_n to a candidate population $\hat{p}_n = \mathcal{G}\left(p_n\right)$, and the second step resamples a new population p_{n+1} from $\hat{p}_n$. The *transition rule* τ is defined as the composition of the two steps.

For the simple genetic algorithm, the heuristic $\mathcal{G}$ can be broken down further into three steps: selection, mutation, and crossover. The *selection scheme* $\mathcal{F}$ maps a population p to a distribution over bit strings that produces an individual x with probability $\mathcal{F}(p, x)$. Significantly, the selection scheme operates at the level of individuals rather than populations. The *mutation function* μ_u^x gives the probability of mutating an individual u to an individual x. A *crossover function* $\chi_z^{x,y}$ *recombines* individual parents x and y into a third individual z in accordance with a crossover rule.

These three operations can be combined to give an explicit form for the action of the heuristic $\mathcal{G}$ on a population p using SGA. Specifically, for each $z \in \{0,1\}^n$, the probability of obtaining an individual z for the next population is given by

$$\mathcal{G}\left(p,z\right)=\sum_{x,y,u,v}\mathcal{F}\left(p,x\right)\mathcal{F}\left(p,y\right)\mu_u^x\mu_v^y\chi_z^{u,v}. \tag{2.5}$$

With some rearrangement, it can be seen that Equation 2.5 is actually a mixture distribution, since it can be rewritten loosely as

$$\mathbb{P}\left(z\in\tau\left(p\right)\right)=\sum_{x\in A}q\left(x\right)\nu^x\left(z\right), \tag{2.6}$$

where A is the set of individuals i such that $\mathcal{F}\left(p,i\right)\neq 0$, $q\left(x\right)\equiv\mathcal{F}\left(p,x\right)$ and

$$\nu^x\left(z\right)=\sum_{y,u,v}\mathcal{F}\left(p,y\right)\mu_u^x\mu_v^y\chi_z^{u,v}. \tag{2.7}$$

This equation makes explicit the claim that mutation and crossover can be represented as a family of probability distributions operating on individuals, and that selection could be viewed as a weighted choice over previously observed individuals. This claim will be used implicitly in Chapter 4 to characterize evolutionary algorithms theoretically.

Here, the heuristic $\mathcal{G}$ is defined on binary search spaces. However, this characterization can be generalized to arbitrary separable measure spaces, which will be done in Chapter 3.

2.7 Quasi-Evolutionary Methods

As discussed in Sections 2.6.1 and 2.1, recent trends in evolutionary computation have focused on developing optimization methods with more detailed mathematical justification. These methods bear strong similarities to evolutionary optimization methods in both terminology and intuitive origins but are not adequately described as attempts at artificial evolution. They are more aptly described as parallelized stochastic optimization techniques with historical ties to evolutionary computation. For these methods, this term *quasi-evolutionary methods* is adopted in this text. In formal terms, it is not possible to distinguish evolutionary and quasi-evolutionary methods completely. Thus this distinction is primarily rhetorical, and will be reflected in the formalisms of Chapters 3 and 4 only to a limited extent.

This section discusses three kinds of quasi-evolutionary methods: differential evolution, estimation of distribution algorithms, and natural evolution strategies. Evolutionary annealing, introduced in Chapter 11 of this dissertation, is a new type of quasi-evolutionary method.

2.7.1 Differential Evolution

Differential evolution was introduced by Storn and Price [198] in 1995 as a means of optimizing the coefficients of Chebyshev polynomials. It has since proven itself as a fast and effective optimizer for finite-dimensional real vector spaces.

Differential evolution maintains a population of candidate solutions that may be thought of as an array of slots. Each slot obeys an independent acceptance-rejection scheme much like in direct search or simulated annealing. At each generation, a new population is created with one candidate solution from each slot. These solutions are evaluated against the objective. The slots are then filled with either the new candidate from that slot or the prior occupant of the slot, depending on which performs better. Thus the objective value of each slot improves monotonically, and differential evolution with population size K consists of K parallel, monotonically improving searches.

The procedure for generating new candidate solutions involves altering one or more components of the real vector that constitutes the solution. Let x be the member of the population in the i^{th} slot. To generate a new solution $\tilde{x}$ from x, the following steps are used. First, three other members of the population are selected, say, a, b, and c. Then for each component j, set $\tilde{x}_j = a_j + F\left(b_j - c_j\right)$ with probability CR and $\tilde{x}_j = x_j$ otherwise. If no component of $\tilde{x}$ is changed from x, then randomly select a component $\tilde{j}$ and apply the change above to that component. The value F is a parameter of the algorithm termed the *weighting factor* that regulates the step size through the search space, and CR is the *crossover rate*, a probability governing the rate at which components are mixed across the population. Typically, $F \in [0, 1]$ with $F = .2$ and $F = .9$ being common values. The crossover rate is also often set to $CR = .2$ or $CR = .9$.

Ghosh et al. [74] showed that differential evolution in the form above converges to the global optimum on continuous real-valued functions that possess a unique global optimum. A number of variants have been developed as well. Some of them move components in the direction of the best member of the population rather than in a random direction (i.e. they replace the vector a above with the best solution in the population). Other methods use crossover operations to further mix the members of the population. Still other methods relax the acceptance criterion to operate more like simulated annealing [52]. Many of these methods have performed well on benchmarks and in optimization competitions, making differential evolution one of the best performing gradient-free stochastic optimizers available today [74].

2.7.2 Estimation of Distribution Algorithms

Evolutionary algorithms can be thought of as building a sequence of probability distributions used to sample each generation. The process begins with an initial distribution that is used to sample the first population. Then, the population is scored, and a new population is created stochastically. The new population is just a sample from some abstract probability distribution. This point of view begs the question: can the optimization algorithm be improved by making this abstract distribution explicit?

Estimation of Distribution Algorithms (EDAs) were the first quasi-evolutionary algorithms to seek to learn a distribution explicitly governing the transition between populations [140, 154]. EDAs rank the prior population according to fitness. The worst members of the population are discarded, and the remaining solutions are used as a dataset to estimate parameters for a probabilistic model, usually some sort of graphical model [122, 140, 152]. Because EDAs arose out of the genetic algorithms research, they are typically applied to objectives with binary encodings, and so multinomial Bayesian networks are a suitable probabilistic model.

Baluja et al. introduced the first EDA, Population-Based Incremental Learning (PBIL), in 1994 [17, 18]. PBIL treats each bit in the encoding as an independently sampled Bernoulli random variable. Despite its simplicity, PBIL can outperform traditional genetic algorithms on several problems with significantly fewer evaluations. Hill Climbing with Learning (HWcL) and the

Compact Genetic Algorithm (cGA) implement the same concept with distinct update rules for the probability vector [86, 120].

Mühlenbein generalized PBIL by considering a variety of approaches that attempt to implement proportional selection statistically [142]. That is, given proportional selection $\mathbb{P}^t_{prop}$ as in Equation 2.4, the algorithm estimates proportional selection by some distribution $\mathbb{Q}^t$ at each generation so that

$$\mathbb{Q}^t\left(x\right) \approx \mathbb{E}\mathbb{P}^t_{prop}\left(x\right), \tag{2.8}$$

where the expectation is taken over populations, that is, over the variable n_x in Equation 2.4. The rationale here is that if the initial population is uniformly distributed, then at time t the pointwise expected value of the proportional selection rule yields

$$\mathbb{E}\left[\mathbb{P}^t_{prop}\left(x\right)\right] \propto f\left(x\right)^t, \tag{2.9}$$

which is a sharpened version of the fitness function. Because of normalization, the result is that as $t \to \infty$, $\mathbb{E}\left[\mathbb{P}^t_{prop}\left(x\right)\right]$ goes to one at the maxima of f and zero elsewhere. Proportional selection in genetic algorithms fails to achieve this effect because a specific sample path is followed rather than the pointwise average. By attempting to model the underlying distribution rather than relying on a sample path, EDAs attempt to benefit from convergence to the optimum. A similar sharpening effect to that observed in Equation 2.9 will be proposed in Chapter 11 without the accompanying context of proportional selection.

A series of subsequent algorithms provided increasingly accurate approximations of Equation 2.9 [140, 142, 157]. The Factorized Distribution Algorithm (FDA) extends this analysis to arbitrary dependencies between variables by estimating $\mathbb{E}\mathbb{P}^t_{prop}$ with a graphical model [122, 140, 152], but FDA does not incorporate a structure learning algorithm. Pelikan introduced the Bayesian Optimization Algorithm (BOA) to provide structure learning details for Bayesian networks and Hierarchical BOA (hBOA) to extend this framework to a hierarchical graphical model [155, 156]. The Real Bayesian Optimization Algorithm (rBOA) translates BOA into an optimization method for real vector spaces [3]. MARLEDA applies similar techniques using a Markov random field rather than a Bayesian network [6].

The class of EDAs thus effectively converts the basic evolutionary algorithm into a probabilistic population-based algorithm that proceeds by matching the structure of a probability distribution to the distribution governing increasingly fit solutions as captured in the mean proportional selection rule of Equation 2.9. It is important to note that the techniques developed for EDAs primarily work for fixed-length binary strings. Evolutionary annealing, introduced in Chapter 11, is similar in some respects to EDAs, but employs mixture distributions that are considerably simpler than Bayesian networks. But evolutionary annealing can be applied naturally to complex domains, and it will be seen that these mixture models approximate more complex models in the limit.

2.7.3 Natural Evolution Strategies

Evolution strategies was mentioned above as a major branch of evolutionary algorithms for real vector spaces; it is reviewed more thoroughly in Chapter 4. Its most common version is characterized by global adaptation of the Gaussian mutation parameters embedded into each candidate solution [27, 165, 183]. In 1996, Hansen and Ostermeier [85] introduced a scheme for adapting the mutation parameters to use elliptical Gaussians with arbitrary rotations and named it Correlated Matrix Adaptation Evolution Strategies (CMA-ES). Over time, the algorithm changed substantially so that rather than storing Gaussian parameters on each candidate solutions, a single global set of Gaussian parameters were used to generate each new generation [84]. These changes occurred around the same time as Estimation of Distribution algorithms were being developed to search binary spaces, and they share important characteristics. The naming of CMA-ES retains the standard $(\mu/\rho + \lambda)$ notation of traditional evolution strategies, but the algorithm itself has few similarities with earlier evolution strategies and little if anything to do with artificial evolution.

In the current version of a (μ, λ)-CMA-ES, a single d-dimensional Gaussian distribution (μ_n, Σ_n) is updated with each generation. The initial population of λ solutions is generated randomly. The population is evaluated, and then the mean and covariance of the best μ solutions are calculated $(\mu < \lambda)$. Then the global Gaussian parameters are updated to incorporate this new information in a manner that smoothly integrates the results of subsequent generations. The details of these parameter updates are complex and can be found in the literature [84]. Because CMA-ES uses a single multivariate Gaussian to generate its population, the search cannot adapt itself to a multimodal objective landscape. In practice, CMA-ES converges to a local optimum relatively quickly, and may be restarted in order obtain good results [13].

Wierstra et al. introduced Natural Evolution Strategies (NES) in 2008, and Akimoto et al subsequently demonstrated that NES is a generalization of the standard CMA-ES algorithm [5, 215]. In NES, the stochastic optimization method is represented as a parameterized probability distribution $\pi\left(x \mid \theta\right)$ where the parameters θ are drawn from a real parameter space and each population samples individuals independently from $\pi\left(x \mid \theta\right)$. This representation can be used to generate a meta-optimization problem of choosing the parameters θ to optimize the expected value of the objective function under π,

$$J(\theta) = \mathbb{E}_\theta\left[f(x)\right].$$

Gradient descent (or ascent) may be applied to this function using

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta\left[f(x)\nabla_\theta \log \pi\left(x \mid \theta\right)\right],$$

which may be estimated by Monte Carlo integration. A final improvement, termed *natural gradient descent* [215], applies a quasi-Newton method replacing the Hessian with the Fischer information matrix, which captures second-order information about how the distribution π changes with the parameters θ.

Natural Evolution Strategies has a firm theoretical foundation and good performance on test problems. Existing versions are limited by the choice of parameterized distribution, which may not align well with the objective being searched. However, this field is relatively new and can be expected to make several useful contributions to the stochastic optimization literature in the near future.

2.8 Conclusion

This chapter has reviewed the primary branches in deterministic and stochastic optimization research, including Newton and quasi-Newton methods, line search, simplicial methods, generating set search, simulated annealing, evolutionary algorithms, natural computation, and quasi-evolutionary methods. It is hoped that the reader has acquired an appreciation for the diversity and scope of these methods. Each of these approaches exists for a distinct purpose because it performs (or used to perform) reliably well on certain kinds of objectives relative to other concurrent methods.

Given the variety of descriptions and terminology among these various optimizers, it may seem challenging to organize all of them within a single formalism. The subsequent chapters seek to accomplish exactly this goal. The key observation is that every method discussed so far produces a sequence of proposed solutions. The formal study of iterative optimization is built on the analysis of this sequence, beginning with the next chapter.

Chapter 3

Functional Analysis of Optimization

In the previous chapter, the major approaches to optimization were reviewed. At a first glance, these approaches present themselves as autochthonous, unrelated, and independent. Indeed, many of these methods seem opposed to each other in both derivation and intent. Evolutionary algorithms appeal single-mindedly to biological analogies. Gradient-based methods blithely assume that local optima will always suffice. These different techniques are alien to each other; it is difficult to see how they can be expressed in a common framework. Yet all of these methods may be compared with each other by analyzing the sequence of solutions each one proposes. A formal analysis along these lines is presented in this chapter.

3.1 Motivation

A germ of similarity among these different methods can be found among these methods in the most basic aspect of their operation. At its core, a black-box optimizer follows a trajectory through the search domain, and then proposes one or more points to add to the trajectory. These points are evaluated against an objective whose internal structure is known only to a limited extent. Thus the black-box optimizer must propose new points to evaluate based only upon the current trajectory and the objective values along that trajectory. This simple fact is held in common among all optimization methods under consideration by the definition of the optimization problem.

From this seed, it will be possible to derive a formal structure that contains all optimization methods. As is necessary, a formal setting so broad as to include so many disparate algorithms will not be able to provide much detailed analysis without further constraints. However, the mere existence of a universally applicable formal setting for optimization should make new insights possible. In fact, the set of all optimizers possesses a surprising degree of universal structure, and this insight is one of the main contributions of this dissertation.

As a brief overview, optimization algorithms are viewed in this theory as consisting primarily of the mechanism by which the current trajectory is to be extended. The space of all such optimizers is a continuous space. Between any two fixed optimization methods on a common search space, there is an infinite number of optimizers that form a spectrum blending the behavior of the original two methods. Furthermore, if the two optimizers at the endpoints are computable, then so is any point along the line connecting them in optimizer space. As a concrete example, suppose one of the endpoints is a hill-climbing search and the other is genetic algorithm with a binary encoding. Then it is meaningful to speak of an optimizer that is exactly halfway between the two, and in fact such an algorithm can be computed simply by flipping a fair coin at each generation to decide whether the hill-climber or the genetic algorithm will be used to construct the next population. Though this result is the most basic of the facts considered in this chapter, it is perhaps the most profound. Despite the vast diversity of optimizers – from line searches to differential evolution, from gradient descent to ant colony optimization – whenever they are used to solve the same problem, they can be smoothly transformed into each other.

With this overview in mind, the following sections develop the basic theory of stochastic optimization. These definitions should aid the reader in identifying the similarities and differences between the wide variety of optimization methods that now exist. The definitions and constructs that follow should also be useful for directing future research towards new and profitable directions.

3.2 Optimizer Space

This dissertation analyzes the structure and behavior of iterative stochastic optimizers on static fitness functions. The first step in this analysis is to define a mathematical space that can be thought of as the space of all optimizers. The study can then proceed by considering the properties of that space using standard analytic techniques and terminology. The next few sections introduce the formal context and define optimizers as mathematical objects.

3.2.1 Assumptions

Every optimization problem begins with a space to be searched and a fitness function to be optimized. In this dissertation, optimization is assumed to be synonymous with minimization; a function f can be maximized by minimizing its additive inverse $-f$. This inquiry into the nature of stochastic optimizers will begin with some assumptions on the nature of the search space and the fitness function.

First, the search space is assumed to be a topological space (X, τ) where X is the collection of possible solutions and τ is a given topology. A topology on a space X is a set of subsets of X that are to be considered as open sets, with the requirement that the empty set and the space X both be in the topology and that finite intersections and countable unions of open sets are also in the topology [144]. Topologies are mainly used to reason about issues such as continuity, limits, and nearness without reference to distance metrics.

In addition, the search space will also be a measurable space $(X, \mathcal{B}_\tau)$, where $\mathcal{B}_\tau$ is the Borel σ-algebra on the topology τ. A σ-algebra on a space X is a set of subsets of X that can be measured [30, 43, 83]. That is, they preserve certain intuitive notions about volume or area. For instance, if any two subsets of X can be measured, then so can their union, intersection, and complements. The σ-algebra is necessary in the definition of a measure space because even in spaces such as the familiar Euclidean space there are subsets for which standard intuitions about volume do not hold up (e.g. additive decomposability over disjoint sets). A Borel σ-algebra is a σ-algebra formed by taking the closure of a topology under countable intersections, unions, and complements; the Borel σ-algebra is the smallest σ-algebra containing the open sets of the topology.

These requirements are quite broad and accommodate all familiar spaces on which optimization is performed, including binary strings, real vectors, neural networks, graphs, state machines, and programs.

The objective function is drawn from the space of real functions on X, denoted $\mathbb{R}^X$. The topology of pointwise convergence is assumed for this function space. Under this topology, a set of functions $\{f_n\}$ converges to a function f if and only if $f_n(x) \to f(x)$ for all $x \in X$. When a σ-algebra on $\mathbb{R}$ is required, the standard Borel σ-algebra for the Euclidean topology is assumed [30, 83]. Occasionally but not often, the objective is required to be Borel-measurable. In this case, the level sets of the objective function must be contained in the Borel σ-algebra.

The formalization below relies heavily on measure theory. A measure is a set function (usually nonnegative) that assigns a volume to each set in a σ-algebra. Lebesgue integration over a real function with respect to a measure sums up the measure of the level sets of the function, i.e

$$\int_X f \, d\mu = \int_{-\infty}^{\infty} \mu\left(\{x : f(x) \leq y\}\right) \, dy.$$

A function f is measurable if the sets $\{x : f(x) \leq y\}$ are contained in $\mathcal{B}_\tau$ for all $y \in \mathbb{R}$. It is integrable on a measure μ if $\int_X |f| \, d\mu < \infty$ [43, 83]. The set of all integrable functions for a particular measure is a complete normed vector space, denoted by $L^1[X, \mu]$. For a given measure μ, the search space is a measure space, written as $(X, \mathcal{B}_\tau, \mu)$. In Euclidean space, μ is assumed to be the Lebesgue measure, the familiar measure of volume. Evolutionary annealing, to be introduced in Chapter 11, requires an explicit measure to be defined for the search space, but most optimization methods do not.

With the formal context of optimization defined in this manner, a mathematical definition of an optimizer can now be stated.

3.2.2 Basic Representation

Stochastic optimization procedures generate successive populations probabilistically with the intent that later populations should contain more optimal solutions. A deterministic optimizer can be regarded as a degenerate stochastic optimizer, and thus deterministic methods are included in this analysis as well.

A stochastic optimization procedure can be described completely by the specification of (1) a (possibly degenerate) distribution over the search space giving the initial evaluation point, (2) a transition rule in the form of a conditional probability distribution over the search space for the next evaluation point given the prior trajectory and its evaluations, and (3) a stopping criterion which decides whether the optimizer should halt given the population history and its fitness scores. To run such an optimizer, the initial trajectory with one point is created by sampling the initial distribution, and its members are scored. Subsequent evaluation points are sampled from the transition rule. When each population is sampled and its fitness evaluated, then the stopping criterion is consulted to decide whether to halt. The output of the optimizer is typically the member of population history with the lowest score on the objective function (recall that minimization is assumed).

To simplify the analysis, the initial distribution will be absorbed into the transition rule. The stopping criterion will be ignored for the time being, although ultimately it will be represented as a stopping time for the purpose of evaluating performance. Thus an optimizer will be described solely by a transition rule. This rule can be identified with the optimizer because the transition rule generates the points used to evaluate the objective function. A similar perspective in a simpler context can be found in Vose's random heuristic search, where Vose's heuristic plays the role of the transition rule [206].

Later in this dissertation, a distinction is made between how an optimizer generates a single point and how it generates an infinite sequence of proposed solutions. When this is done, the optimizers described in this chapter are referred to as *one-step optimizers*, as opposed to the *long-running* or *extended* optimizers that result from running one-step optimizers iteratively to generate a sequence. The two objects are closely related, and their relationship is discussed thoroughly in Chapter 6.

Consider the identity of a optimizer as a mathematical object. At a general, abstract level, an optimization method is a function that maps a trajectory along with its evaluations and attendant information (such as derivatives, errors, and other side effects of evaluation) into a probability measure over populations. That is, for each particular trajectory (including the empty trajectory) and each particular objective function, an optimizer must specify a probability measure that can be sampled to produce the next point for evaluation.

Evolutionary methods generate entire populations, not individual points, so a probability distribution over individual points may seem inappropriate. However, each population is nothing more than a collection of points. A population can be generated by sampling the new set of points one at a time with the appropriate dependency relationships. While it may seem strange to discuss evolutionary algorithms in terms of sampling a probability distribution, it is entirely correct, since the underlying probability distribution is nothing other than the mechanism by which the next population is created from previous populations with random variations. An alternative approach is to treat evolutionary methods as operating on the search space X^K, so that a evaluation point is an entire population, with an altered objective function $f' : X^K \to \mathbb{R}^K$, but this perspective obscures the relationship to other optimization methods as well as the internal relationship between instances of an evolutionary algorithm with different population sizes. Upon consideration, the one-point-at-a-time analysis is found to be the more elegant representation.

From here on, a stochastic optimizer $\mathcal{G}$ is a function $\mathcal{G}[t, f]$ that takes a finite trajectory and its objective evaluations as inputs and returns a probability distribution as outputs. To formalize this functional space, both the domain and range of these functions must be specified. For the domain, let $\mathcal{T}[X]$ be the space of all sequences on the search space X of finite but arbitrary length, so that an element in $\mathcal{T}[X]$ is a finite trajectory of candidate solutions. For the objective function, any real function over the search domain will be allowed. That is, the objective f is an element of $\mathbb{R}^X$, the space of arbitrary real functions on X. This choice implies that the objectives under consideration are static real functions. Let $\mathcal{P}[X] = \mathcal{P}[X, \mathcal{B}_\tau]$ be the space of probability measures on $(X, \mathcal{B}_\tau)$. Then the set of optimizers is contained in the set $\mathcal{PF}[X, \mu] \equiv \left\{\mathcal{G} : \mathcal{T}[X] \times \mathbb{R}^X \to \mathcal{P}[X]\right\}$, where $\mathcal{PF}$ stands for *probability-valued functions*.

It might seem odd to refer to members of $\mathcal{PF}$ as optimizers, since they are in fact merely functionals that output probability distributions. However, it is no exaggeration to say that every single member of $\mathcal{PF}$ is in fact an optimizer in the sense defined above. Let $\mathcal{G} \in \mathcal{PF}$ be arbitrary. Then $\mathcal{G}[t, f]$ is a probability distribution for each t, f. An optimizer does not resample points if with probability 1, $\mathcal{G}$ produces a new point not in t, i.e. $\mathcal{G}[t, f](z \in t) = 0$.

If $\mathcal{G}$ does not resample, then the No Free Lunch theorem suggests that as long as $\mathcal{G}$ relies only on the evaluations of f on t, $\mathcal{G}$ must work better than other optimizers on some objective [218]. Thus $\mathcal{PF}$ is a suitable space in which to begin an exploration of stochastic optimization.

3.2.3 Notation and Conventions

This subsection introduces notation and conventions that are used throughout subsequent chapters. Optimizers will typically be denoted by capital cursive letters, usually by $\mathcal{G}$. The expression $\mathcal{G}\left[t,f\right]$ will be used to refer to the probability measure corresponding to a trajectory $t \in \mathcal{T}\left[X\right]$ and a fitness function $f \in \mathbb{R}^X$. Accordingly, $\mathcal{G}\left[t,f\right](A)$ indicates the probability that the next point will lie inside of a set A contained in the σ-algebra $\mathcal{B}_\tau$. The notation $\mathcal{G}\left[t,f\right](dx)$ represents a quantity that can be integrated over A in the Lebesgue sense to obtain $\mathcal{G}\left[t,f\right](A)$. Occasionally, the space of objectives will be restricted to functional spaces smaller than $\mathbb{R}^X$, such as $L^1[X,\mu]$, the space of μ-integrable functions. In the definition of spaces, the particular search domain or other parameters are often omitted if understood, so that $\mathcal{PF} = \mathcal{PF}\left[X,\mu\right]$ or $L^1 = L^1\left[X,\mu\right]$ and so on. From here on, the term *optimizer* will mean an element in the set $\mathcal{PF}$ or related spaces, although the term may sometimes refer more generally to elements of $\mathcal{MF}$ defined below as required by context. Thus several objects in these spaces will be referred to as optimizers, even though these objects would perform quite poorly at most common optimization tasks.

In later chapters, indicator functions will be used in some definitions and proofs. An indicator function is represented in this dissertation by $1_A(x)$ for a set A; this function is equal to 1 if $x \in A$, and zero otherwise.

Objective functions $f \in \mathbb{R}^X$ are assumed to have a finite minimum, denoted by $f^* = \inf_X f(x) > -\infty$. Obviously, the space $\mathbb{R}^X$ may contain functions that are not bounded below on X. When objectives are referred to specifically, these functions are ignored. Later chapters will place a measure over all of $\mathbb{R}^X$; unbounded functions will be specifically excluded at that point.

As the definitions suggests, only static objective functions are considered in this theory. The formalism could be expanded to accommodate either dynamic or stochastic objective functions, but these adaptations would yield distinct results and would complicate the discussion that follows. Unless mentioned or otherwise clear from context, the terms *optimization* and *optimum* should be interpreted as *minimization* and *minimum* for the sake of understanding the formulae. Additionally, objective functions may be referred to as fitness functions with equivalent meaning, and the fitness of a point $x \in X$ is just its objective value $f(x)$. Points in the search space will be referred to interchangeably as points, individuals, evaluation points, solutions, candidate solutions, proposed solutions, or even organisms with equivalent meaning.

Elements in $\mathcal{T}[X]$ are referred to as evaluation histories. An evaluation history $t \in \mathcal{T}[X]$, also termed a trajectory, an evaluation trajectory, or an evaluation sequence, is a finite list of points. The empty history is required to be in $\mathcal{T}[X]$ and will be denoted by $\emptyset$. As the narrative develops, there will be a need to complete $\mathcal{T}[X]$ to the sequence space $X^{\mathbb{N}}$ containing all countable sequences on X. Elements in $\mathcal{T}[X]$ may be identified with subsets of $X^{\mathbb{N}}$ that share a common finite prefix. Occasionally, an integral is taken over a set $T \subseteq \mathcal{T}[X]$. When this is done, the integration is actually to be performed over the set of all elements in $X^{\mathbb{N}}$ that extend any member of T.

Trajectories will be indexed using superscripts, so that t^n indicates the n^{th} evaluation point in t, with indices starting at 1 for the initial point, with $t^0 = \emptyset$. Negative superscripts index the trajectory backwards, so that t^{-1} is the last point in t, t^{-2} the next to last, and so on. Subscripts on trajectories indicate a sequence of trajectories, so that e.g. t_n is not a point, but an entire sequence of points. Thus t_n^m represents a particular point within a sequence of population histories. Two trajectories can be concatenated to form a longer trajectory, denoted by a union operator, e.g. $t = t_1 \cup t_2$. An element $x \in X$ can also be appended to a trajectory, denoted similarly by $t = t_1 \cup x$. The notation $t \cup \left(\bigcup_{i=1}^K x_i \right)$ indicates successive concatenation, i.e. $t \cup x_1 \cup x_2 \cup \cdots \cup x_K$. Given a sequence $(x_n)_{n=1}^{\infty}$, the expression $(x_n)_{n=1}^{N}$ represents a trajectory of length N, and $(x_n)_{n=1}^{0} = \emptyset$ by convention. In addition to indexing, the notation $x \in t$ will be used to indicate that x is an arbitrary point occurring at some point in t, i.e., $x = t^n$ for some n.

The notation $H(t)$ is used to convert a trajectory in $\mathcal{T}[X]$ to a trajectory in $\mathcal{T}[X^K]$ for some fixed K that will be clear from the context. When this is done, $H(t)^n$ refers to the n^{th} entry of $H(t)$, an element of X^K, and $H(t)^{n,k}$ refers to the k^{th} component of the n^{th} entry, an element of X. This mapping is further described in Chapter 4.

Consider the process of running an optimizer $\mathcal{G} \in \mathcal{PF}$ on an objective function f. First, the trajectory is initialized to $t_0 = \emptyset$. Then, a point $x_1 \in X$ is sampled from $\mathcal{G}\left[t_0, f\right]$. This population is appended to t_0 to create $t_1 = t_0 \cup x_1$. Next, a population x_2 is sampled from $\mathcal{G}\left[t_1, f\right]$ and appended to t_1 to form $t_2 = t_1 \cup x_2$. The process continues until a stopping criterion is reached. Thus in actual practice the trajectory t is progressively sampled from the optimizer $\mathcal{G}$, and the trajectory takes on random values. This random sequence of evaluation points is a stochastic process, termed the *optimization process*. Given an optimizer $\mathcal{G}$ and an objective f, the notation $\mathcal{G}_f$ or $\mathcal{G}f$ will be used equivalently to represent the distribution of this random process. $\mathcal{G}_f$ is a distribution over $\mathbb{X}^{\mathbb{N}}$, the space of infinite sequences on the search space. The existence of $\mathcal{G}_f$ is discussed in Chapter 6. Any property that holds for a set of sequences with full measure in $\mathcal{G}_f$ is said to hold $\mathcal{G}_f$-*a.s.* If a property of trajectories in $\mathcal{T}[X]$ holds for all prefixes of such a set, then that property is also said to hold $\mathcal{G}_f$-*a.s.*

The optimization process will be denoted by $Z = (Z_n)_{n\in\mathbb{N}}$. Any process Z that is distributed according to $\mathcal{G}_f$ is said to be *generated by* $\mathcal{G}$ on f, also written as $Z \sim \mathcal{G}_f$. The natural filtration of the optimization process will be written as $\{\mathcal{Z}_m\}$ The evaluation point corresponding to the running minimum of the optimization process for a particular objective will be denoted by $Z^* = (Z^*_n)_{n\in\mathbb{N}}$. The optimization process will be used extensively to define performance criteria for assessing the performance of optimizers. Chapter 6 explores the optimization process in further detail.

3.2.4 Information Restrictions

The set $\mathcal{PF}$ contains all iterative stochastic optimizers, including many that are uncomputable. Optimizers in $\mathcal{PF}$ have full, direct access to the objective function. For example, suppose two different fitness functions f and g are equal on a given trajectory but have distinct global optima. Then no optimizer should be able to distinguish between them on the basis of that trajectory. But $\mathcal{PF}$ is largely composed of optimizers that do indeed make such a distinction, and therefore $\mathcal{PF}$ does not capture the primary intuitions about how a stochastic optimizer should work. These intuitions can be restored by defining a property that characterizes optimizers that only consider the fitness evaluations of the population history and do not distinguish between functions that are equal on a given population history. Such an optimizer will be termed *trajectory-restricted*:

Definition 3.2.1. *An optimizer $\mathcal{G}$ is trajectory-restricted if $\mathcal{G}[t,f] = \mathcal{G}[t,g]$ whenever $f(x) = g(x)$ for every $x \in X$ appearing in t.*

Let $\mathcal{O}_{\text{tr}}[X,\mu]$ be the subset of $\mathcal{PF}[X,\mu]$ such that every element in $\mathcal{O}_{\text{tr}}$ is trajectory-restricted. Elements of $\mathcal{O}_{\text{tr}}$ can only use information obtained from evaluations of the function. This fact excludes gradient-based optimizers, which can distinguish functions with equivalent fitness evaluations if they have different gradients. However, it does not exclude optimizers that use an estimated gradient computed from the function evaluations. Also, the vast majority of evolutionary and Monte Carlo methods are trajectory-restricted.

Nonetheless, whereas $\mathcal{PF}$ is too large, $\mathcal{O}_{\text{tr}}$ is too small; it excludes methods that use information other than just the function evaluations. In order to include gradient-based methods *inter alia*, optimizers must be allowed to receive information from the function evaluation that can then be fed into the gradient function. Indeed, gradient methods are not the only optimizers that receive information from the objective function. Expectation maximization proposes new model parameters based on the statistics of its current model and may not even evaluate the current estimated log likelihood. Even some evolutionary methods, such as novelty search, make use of statistics gathered during function evaluation [125]. The information used by these algorithms can be realized as a finite trajectory over one-dimensional Euclidean space, that is, as an element of $\mathcal{T}[\mathbb{R}]$. Such a trajectory can be a sequence of error signals or a fixed set of statistics, or whatever else is required. Each optimizer that makes use of such information would then be associated with a function $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$ so that $I(f,x)$ is the desired information signal.

Definition 3.2.2. *An optimizer $\mathcal{G}$ is information-restricted if there exists an information signal $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$ such that $\mathcal{G}[t,f] = \mathcal{G}[t,g]$ whenever $I(f,x) = I(g,x)$ for every $x \in X$ appearing in t.*

Let $\mathcal{O}_{\mathrm{ir}}[X, \mu]$ be the subset of $\mathcal{PF}[X, \mu]$ such that each optimizer in $\mathcal{O}_{\mathrm{ir}}$ is information-restricted. Then $\mathcal{O}_{\mathrm{ir}}$ is the set of optimizers that rely on evaluation-dependent information. The particular function I associated with an optimizer $\mathcal{G}$ is termed its *information function.* Because each optimizer is allowed to select the information it will require, the particular information function varies with each information-restricted optimizer.

The class of information-restricted optimizers is a superset of the class of trajectory-restricted optimizers, since for any $\mathcal{G} \in \mathcal{O}_{\mathrm{tr}}$ the information function $I(f, x) = f(x)$ makes $\mathcal{G}$ information-restricted as well, that is, $\mathcal{O}_{\mathrm{tr}} \subseteq \mathcal{O}_{\mathrm{ir}}$. Unfortunately, the information function I does not lend itself to easy analysis, and without further restrictions on I, an optimizer could craft I to evaluate a whole series of points in addition to x. Since the eventual goal is to compare optimizers based on the number and outcome of evaluations, the opacity of the information function I will introduce complications. Most but not all of the analysis in this dissertation will pertain only to $\mathcal{O}_{\mathrm{tr}}$ rather than the larger $\mathcal{O}_{\mathrm{ir}}$. However, when possible, results will also be given for $\mathcal{O}_{\mathrm{ir}}$.

3.2.5 Computability of Optimizers

Neither information restrictedness nor trajectory restrictedness can account for perhaps the most important practical consideration: computability. In order to keep the discussion focused, some common details of computability will be ignored. The theory works with $\mathbb{R}$ and other infinite spaces directly, even though in practice elements of these spaces cannot be represented in a finite and discrete computer. It will be assumed that reasonable approximations such as floating point numbers are used for $\mathbb{R}$, and fixed elements of the search space X will be assumed to have a workable finite and discrete representation.

In determining what makes an optimizer computable, one need only consider computable objective functions, since uncomputable objectives will render computable optimization impossible. An objective function f is computable if there exists a Turing machine that takes a representation of any element $x \in X$ as an input and halts with $f(x)$ on its tape. The objective function is polynomially (or exponentially) computable if it is computed by a Turing machine that halts in time polynomial (or exponential) in the size of

x. An information function I (as in the prior section) is computable if there is a Turing machine for I that, given any computable objective function f, takes as input a Turing machine that computes f and an element $x \in X$ and halts with $I(f, x)$ on its tape. An information function is polynomially (or exponentially) computable if it is computed by a Turing machine that halts in time polynomial (or exponential) in the size of the input. Notice that the computability of an information function is defined based only on computability with respect to computable inputs; the same principle will be required of a computable optimizer. Only information-restricted optimizers will be considered for this purpose. These definitions of computable functions are standard in computation theory [188].

Before giving a definition for a computable optimizers, it is necessary to consider what an optimizer computes. Optimizers as defined here produce a trajectory in the search space one point at a time by sampling a probability distribution. Thus computability of an optimizer is equivalent to the computability of the sampling operation. The following definitions are introduced here in the spirit of standard computation theory.

Definition 3.2.3. *A Turing machine M approximately samples a probability distribution $\mathbb{Q}$ over a measure space $(X, \mathcal{F})$ if, given error $\epsilon > 0$ as input, M halts in finite time with an element $x \in X$ on its tape, and if for any $A \in \mathcal{F}$, $|\mathbb{P}_M(x \in A) - \mathbb{Q}(A)| < \epsilon$. M is called a polynomial (or exponential) sampler if it halts in time polynomial (or exponential) in $1/\epsilon$.*

Definition 3.2.4. *An information-restricted optimizer $\mathcal{G} \in \mathcal{O}_{\mathrm{ir}}[X, \mu]$ is computable if its information function I is computable and if there exists a Turing machine M such that when M is given a Turing machine that computes I, a Turing machine that computes an objective f, and a finite trajectory $t \in \mathcal{T}[X, \mu]$, then M halts and outputs a second Turing machine M' that approximately samples $\mathcal{G}[t, f]$. The optimizer $\mathcal{G}$ is polynomially (or exponentially) computable (1) if there exists a Turing machine M that computes it in time polynomial in the size of the representation of t and the size of the Turing machines for I and f, and (2) if the output M' of M is a polynomial (or exponential) sampler for all inputs to M.*

Although the definition requires a good deal of text, its intent is simple. A computation procedure for an optimizer $\mathcal{G}$ requires a representation of the objective f, a representation of the information function, and a trajectory t of previously evaluated points. Given these items, the procedure produces a module that can sample from $\mathcal{G}\left[t, f\right]$ within a given tolerance ϵ. The optimizer $\mathcal{G}$ is computable if all of its parts are, and its efficiency is evaluated with respect to the size of its inputs and the stringency of the tolerance.

Now let $\mathcal{O}_{\text{ir}}^{\text{co}}$ be the set of all computable information-restricted optimizers, and similarly for $\mathcal{O}_{\text{tr}}^{\text{co}}$. Let $\mathcal{O}_{\text{ir}}^{\text{poly}}$ be the set of all polynomially computable information-restricted optimizers, and again similarly for $\mathcal{O}_{\text{tr}}^{\text{poly}}$.

Note that the distinction between polynomially computable and generally computable optimizers is an important one. While it is tempting to disregard optimizers that are not polynomial, several interesting optimizers in the literature are not polynomial. For instance, the Bayesian Optimization Algorithm, which builds successive Bayesian networks that model the correlations among the most successful evaluation points, is necessarily non-polynomial, because even a greedy structure search in a Bayesian network is non-polynomial. Other optimizers, such as curiosity search [177] or some instances of expectation maximization, require an internal optimization loop. These techniques may be non-polynomial if the internal optimization is non-polynomial or is invoked exponentially many times. A non-polynomial optimizer can still be feasible to run provided that the inputs are of small to moderate size. Thus non-polynomial optimizers should not be disregarded.

It is also important to note that the spaces of computable optimizers can be much smaller than corresponding spaces that include non-computable optimizers. A computable optimizer must have a representation as a finite program, and there are only countably many finite programs. By contrast, the cardinality of spaces like $\mathcal{O}_{\text{tr}}$ is typically much larger.

With the basic spaces of optimizers defined, we turn now to consider how these optimizers may be employed for the purpose of optimization, and what general operators are available on this space.

3.3 Algebraic Operations

Optimizers can be combined or altered algebraically to form a new optimizer in several ways. In this section, some mechanisms for algebraically combining operators are discussed.

3.3.1 Convolution

The first operator will be termed convolution due to its similarity to the convolution of two functions. In this case, the two optimizers being convolved represent the substructure of a third optimizer, and may or may not be practical optimizers on their own. The convolution operator, denoted by $\star$, is defined by the equation

$$(\mathcal{G}_1 \star \mathcal{G}_2)\left[t, f\right](A) = \int_X \mathcal{G}_2\left[t \cup x, f\right](A)\;\; \mathcal{G}_1\left[t, f\right](dx)\,. \tag{3.1}$$

Convolution performs the intuitive function of applying two probability distributions in sequence. First, a point is sampled from $\mathcal{G}_1$, and then a point is sampled from $\mathcal{G}_2$ given the outcome of sampling $\mathcal{G}_1$. In fact, the entire process of stochastic optimization described in the previous sections boils down to the successive application of the convolution operator, so that if (Z_n) is generated by $\mathcal{G}$, then (Z_{2n}) is generated by $\mathcal{G} \star \mathcal{G}$. It is even possible to write $Z_n \sim \left(\bigstar_{m=1}^n \mathcal{G}\right)\left[\emptyset, f\right]$ when $Z \sim \mathcal{G}_f$, where $\bigstar_{m=1}^n \mathcal{G}$ represents n successive applications of convolution. When two or more convolution operators are used, convolution is assumed to be left associative, e.g. $\mathcal{G}_1 \star \mathcal{G}_2 \star \mathcal{G}_3 = (\mathcal{G}_1 \star \mathcal{G}_2) \star \mathcal{G}_3$. Because convolution is not necessarily commutative, right association is not equal to left association, and so the postfix notation $(\mathcal{G} \bigstar_{m=1}^n)$ will indicate chained right associations, e.g. $\mathcal{G}_1 \star (\mathcal{G}_2 \star \mathcal{G}_3)$. The description of evolutionary algorithms in particular can be substantially simplified by the use of the convolution operator.

The convolution of two computable optimizers is computable. The convolution of two polynomial optimizers is polynomial. Polynomially many convolutions of polynomial optimizers can be done in polynomial time. More interestingly, convolution does not preserve information-restrictedness or trajectory-restrictedness in general. When two information-restricted optimizers are convolved, then the internal point x in Equation 3.1 is hidden from the algorithm's progress. Consider the case of convolving two trajectory-restricted optimizers $\mathcal{G}_1$ and $\mathcal{G}_2$. To compute the convolution, a point x is sampled first from $\mathcal{G}_1[t, f]$, and then a new point y is sampled from $\mathcal{G}_2\left[t \cup x, f\right]$. Then, the next point will be sampled from $\left(\mathcal{G}_1 \star \mathcal{G}_2\right)\left[t \cup y, f\right]$ with the point x suppressed. Thus if $f(x) \neq g(x)$, then it is possible that $\left(\mathcal{G}_1 \star \mathcal{G}_2\right)\left[t, f\right] \neq \left(\mathcal{G}_1 \star \mathcal{G}_2\right)\left[t, g\right]$ even if $f(y) = g(y)$ for all $y \in t$. Therefore $\mathcal{G}_1 \star \mathcal{G}_2$ is not trajectory-restricted. A similar line of reasoning holds for information-restricted optimizers.

Since convolutions will be used to construct evolutionary algorithms explicitly, and because evolutionary algorithms are generally trajectory-restricted, it is worth it to consider when a convolution may be trajectory-restricted. The simplest way to preserve the trajectory restriction is to disallow evaluation of the internal points. Such an optimizer will be termed *objective-agnostic*:

Definition 3.3.1. *An optimizer $\mathcal{G} \in \mathcal{PF}$ is objective-agnostic if $\mathcal{G}\left[t, f\right] = \mathcal{G}\left[t, g\right]$ for all $f, g \in \mathbb{R}^X$.*

Proposition 3.3.1. *If $\mathcal{G}_1$ is a trajectory- (or information-) restricted optimizer and $\mathcal{G}_2$ is an objective-agnostic optimizer, then $\mathcal{G}_1 \star \mathcal{G}_2$ is trajectory- (or information-) restricted.*

Proof. Let $t \in \mathcal{T}$, $f, g \in \mathbb{R}^X$ with $f(x) = g(x)$ for all $x \in t$. Let $\mathcal{G}_1 \in \mathcal{O}_{\text{tr}}$ and let $\mathcal{G}_2$ be objective-agnostic. Let $\mathcal{G} \equiv \mathcal{G}_1 \star \mathcal{G}_2$. Then

$$\begin{aligned}
\mathcal{G}[t, f](A) &= \int_x \mathcal{G}_2[t \cup x, f](A)\mathcal{G}_1[t, f](dx) && (3.2)\\
&= \int_x \mathcal{G}_2[t \cup x, g](A)\mathcal{G}_1[t, g](dx) && (3.3)\\
&= \mathcal{G}[t, g](A) && (3.4)
\end{aligned}$$

The conclusion for information-restricted optimizers follows by choosing $\mathcal{G}_1 \in \mathcal{O}_{\text{ir}}$ with information function I_1. Then choose $f, g \in \mathbb{R}^X$ so that $I_1(f, x) = I_1(g, x)$ for all $x \in t$. Repeating the same equations as above shows $\mathcal{G} \in \mathcal{O}_{\text{ir}}$ with information function I_1. □

Notice that an objective-agnostic optimizer is trivially trajectory-restricted and information-restricted, since it cannot depend on a single evaluation of the objective function. However, it can depend on the trajectory. The mutation operators for evolutionary algorithms that will be defined in Chapter 4 are all objective-agnostic.

Objective-agnostic optimizers are not the only case in which information-restrictedness can pass through convolution. The most general case occurs when the second optimizer is agnostic to the objective only on the last step of the trajectory.

Definition 3.3.2. *An optimizer* $\mathcal{G} \in \mathcal{PF}$ *is said to be one-step objective-agnostic if for all* $x \in X$*, all* $t \in \mathcal{T}$*, and all* $f, g \in \mathbb{R}^X$*,* $\mathcal{G}[t \cup x, f] = \mathcal{G}[t \cup x, g]$ *whenever* $\mathcal{G}[t, f] = \mathcal{G}[t, g]$*.*

Proposition 3.3.2. *If* $\mathcal{G}_1$ *and* $\mathcal{G}_2$ *are both trajectory- (or information-) restricted optimizers, and* $\mathcal{G}_2$ *is also one-step objective-agnostic, then* $\mathcal{G}_1 \star \mathcal{G}_2$ *is trajectory- (or information-) restricted.*

Proof. Repeat the proof of Proposition 3.3.1 *mutatis mutandis.* $\square$

Recombination operators in genetic algorithms will be constructed as one-step objective-agnostic optimizers in Chapter 4.

3.3.2 Trajectory Truncation

Define trajectory truncation by the symbol $\triangleleft$ so that

$$(\triangleleft \mathcal{G})\left[t \cup x, f\right] = \mathcal{G}\left[t, f\right], \tag{3.5}$$

with the base case $(\triangleleft \mathcal{G})\left[\emptyset, f\right] = \mathcal{G}\left[\emptyset, f\right]$. This operator can be applied to the same optimizer more than once. Let $\triangleleft_k \mathcal{G}$ represent the optimizer resulting from $k \geq 0$ applications of trajectory truncation, with $\triangleleft_0 \mathcal{G} \equiv \mathcal{G}$. Notice that the result of trajectory truncation is always one-step objective-agnostic. The trajectory truncation operator will be used extensively as part of the formalization of population-based optimizers. Discussion of further tools for handling population-based optimizers is deferred to Chapter 4.

3.3.3 Convex Combination

Optimizers can be combined convexly to form new operators using the basic operations of pointwise addition and pointwise scalar multiplication. Used by themselves, these two operations are not closed on $\mathcal{PF}$, but their convex combinations are closed. Define pointwise scalar multiplication so that $(\alpha\mathcal{G})\,[t,f]\,(A) \equiv \alpha\,(\mathcal{G}\,[t,f]\,(A))$ for $\alpha \in \mathbb{R}$. Then it is clear that $\alpha\mathcal{G}$ is not a member of $\mathcal{PF}$ for $\alpha \neq 1$, since $\alpha\mathcal{G}[t,f](X) = \alpha$ and so $\alpha\mathcal{G}[t,f]$ is not a probability distribution.

Define pointwise addition so that $(\mathcal{G}_1 + \mathcal{G}_2)\,[t,f](A) \equiv \mathcal{G}_1[t,f](A) + \mathcal{G}_2[t,f](A)$. Again, it is clear that $\mathcal{G}_1 + \mathcal{G}_2 \notin \mathcal{PF}$, but the operation is well-defined nonetheless. Pointwise addition and pointwise scalar multiplication are closed on a larger space that will be examined shortly.

There is a case in which these operations can be used to form a new optimizer in $\mathcal{PF}$. Let $\alpha \in [0,1]$ and consider $\mathcal{G} = \alpha\mathcal{G}_1 + (1-\alpha)\,\mathcal{G}_2$. Then $\mathcal{G}\,[t,f]$ is always a probability distribution, so $\mathcal{G} \in \mathcal{PF}$. More generally, choose $\alpha_1, \ldots \alpha_n$ in $[0,1]$ such that $\sum_i \alpha_i = 1$, and suppose that $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are optimizers. Then $\mathcal{G} = \sum_i \alpha_i \mathcal{G}_i$ is a convex combination of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, and $\mathcal{G} \in \mathcal{PF}$. So $\mathcal{PF}$ is closed under convex combination.

Convex combinations preserve the four optimizer properties introduced so far: computability, computational complexity, trajectory-restrictedness and information-restrictedness. Convex combinations are mixture distributions over optimizers. To sample a convex combination formed by $\mathcal{G} = \sum_i \alpha_i \mathcal{G}_i$, first sample the probability vector α to select the index i. Then sample $\mathcal{G}_i\,[t,f]$, and the result is a sample from $\mathcal{G}\,[t,f]$. Since sampling a probability vector is polynomially computable (to a suitable approximation error), $\mathcal{G}$ will be computable if and only if $\mathcal{G}_i$ is computable for all i such that $\alpha_i > 0$, and $\mathcal{G}$ will be polynomial if the $\mathcal{G}_i$ are. Similar arguments show that information-restrictedness and trajectory-restrictedness are preserved as well. Thus convex combination is closed over these four properties.

To emphasize, $\mathcal{PF}$, $\mathcal{O}_{\text{ir}}$, $\mathcal{O}_{\text{tr}}$, $\mathcal{O}^{\text{co}}_{\text{ir}}$, $\mathcal{O}^{\text{co}}_{\text{tr}}$, $\mathcal{O}^{\text{poly}}_{\text{ir}}$, and $\mathcal{O}^{\text{poly}}_{\text{tr}}$ are all convex spaces, i.e. each of these spaces are closed under convex combinations. For any $\mathcal{G}_1, \ldots, \mathcal{G}_n$ contained in any one of these spaces, all convex combinations also lie inside the same space.

Given two optimizers, the set of all their convex combinations forms a line that blends smoothly between them. Such a line exists between any two optimizers in $\mathcal{PF}$. For any two optimizers in one of the convex subspaces of $\mathcal{PF}$, the line between these optimizers does not leave the subspace at any point.

In actual usage, a convex combination of optimizers can be viewed as a sequence of choices among the combined optimizers. Let $\mathcal{A}, \mathcal{B} \in \mathcal{PF}$, and let $\mathcal{C} = \alpha\mathcal{A} + (1-\alpha)\mathcal{B}$ for a fixed $\alpha > 0$. Then a *history* of $\mathcal{C}$ is the sequence of choices made by $\mathcal{C}$ at each time step. This may be written as e.g. $\mathcal{AAABABBB}\dots$, and the set of all histories of $\mathcal{C}$ may be regarded as the set of optimization strategies available to $\mathcal{C}$. This conception of convex combinations of optimizers evokes game theory, and in fact optimization using multiple optimizers may be regarded as a game-theoretic game played with goal of optimizing the optimization process. This set of ideas will be explored more thoroughly in Chapter 10.

3.4 Measure-Valued Operators: a Normed Vector Space

Pointwise scalar multiplication and pointwise addition are vector operations. They satisfy the standard requirements for vector operations, namely, commutativity and invertibility of addition, the existence of an identity, and the distributivity of multiplication over addition. Thus optimizers in $\mathcal{PF}$ are vectors, but in what vector space? In this section, it will be shown that $\mathcal{PF}$ is a closed, convex subset of a normed vector space, and computable, information-restricted, and trajectory-restricted optimizers are likewise closed, convex sets inside of vector subspaces of this vector space. Convex combinations have already been discussed briefly. The existence of the norm provides a context for approximating one optimizer by a sequence of optimizers. Thus the structures discussed here are not superfluous; they make it possible to think about optimization in a new way.

In order to define the vector space containing $\mathcal{PF}$, consider first the space of finite signed measures. Such a measure is a set-valued function defined over a σ-algebra that is additive on disjoint sets. It may take on both positive and negative values, but must be finite on every set in the σ-algebra. Denote by $\mathcal{M}[X] = \mathcal{M}[X, \mathcal{B}_\tau]$ the space of all finite signed measures on $(X, \mathcal{B}_\tau)$. The space $\mathcal{M}[X]$ is a Banach space, a complete, normed vector space. [1] The standard norm for $\mathcal{M}[X]$ is the total variation norm, given as the largest absolute measure assigned to any set in the σ-algebra, $||\mu||_{\mathcal{M}} \equiv \sup_{A \in \mathcal{B}_\tau} |\mu(A)|$.

3.4.1 The Normed Vector Space $\mathcal{MF}$

The space of probability measures $\mathcal{P}[X]$ on $(X, \mathcal{B}_\tau)$ has already been encountered. $\mathcal{P}[X]$ is a closed, convex subset of $\mathcal{M}[X]$, with the implication that probability measures are vectors that can be added and subtracted or convexly combined. To see this, note that the limit of probability measures is a probability measure (implying that the set is closed) and that any convex combination of probability measures is a probability measure. It should be noted that $\mathcal{P}[X]$ is only a subset and not a vector subspace of $\mathcal{M}[X]$, since the pointwise sum of two probability measures is not a probability measure. Although all probability measures have a total variation norm of 1 by definition, the difference of two probability measures is well defined, non-trivial, and exists in $\mathcal{M}[X]$. This difference defines a distance metric on probability measures, given by

$$d(\mathbb{P}, \mathbb{Q}) = ||\mathbb{P} - \mathbb{Q}||_{\mathcal{M}} = \sup_{A \in \mathcal{B}_\tau} |\mathbb{P}(A) - \mathbb{Q}(A)| \tag{3.6}$$

for probability measures $\mathbb{P}$ and $\mathbb{Q}$. Intuitively, the distance between two probability measures is determined by locating the set to which the two measures assign the largest difference in probability mass and taking the absolute difference in probability between the two on that set.

Now define the functional space

$$\mathcal{MF} = \mathcal{MF}[X, \mu] = \left\{ \mathcal{G} : \mathcal{T}[X] \times \mathbb{R}^X \to \mathcal{M}[X] \right\}. \tag{3.7}$$

The space $\mathcal{MF}_0$ contains $\mathcal{PF}$, but it also contains many other objects as well. An element in $\mathcal{MF}_0$ is a function that produces a finite signed measure over the

---

[1] A vector space provides a high degree of structure, including vector addition and subtraction as well as scalar multiplication. A norm assigns an absolute magnitude to each element in the space and can be used to generate a distance metric. A space is complete if it contains all of its limit points.

search space when given any finite trajectory and any objective function. This space will serve as the basic vector space from which subspaces and subsets of optimizers will be carved out. Define vector operations in $\mathcal{MF}_0$ pointwise as for $\mathcal{PF}$, i.e. for $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{MF}_0$ and $\alpha \in \mathbb{R}$,

$$\begin{aligned}
\left(\mathcal{G}_1 + \mathcal{G}_2\right)[t, f] &= \mathcal{G}_1[t, f] + \mathcal{G}_2[t, f], &(3.8)\\
\left(\alpha\mathcal{G}_1\right)[t, f] &= \alpha\left(\mathcal{G}_1[t, f]\right), &(3.9)
\end{aligned}$$

where vector addition and scalar multiplication on the right are drawn from the vector space structure of $\mathcal{M}[X]$. Because of this, the vector operations satisfy the required associative and distributive properties. The zero vector for $\mathcal{MF}_0$ is the function that returns the zero measure on all inputs. So $\mathcal{MF}_0$ is a vector space. In fact, $\mathcal{MF}_0$ is just the closure of $\mathcal{PF}$ under the operations of pointwise scalar multiplication and addition.

The next step is to find a norm for $\mathcal{MF}_0$ so that the distance between any two optimizers can be compared. A norm can be created from

$$||\mathcal{G}||_{\mathcal{MF}} = \sup_{t \in \mathcal{T}, f \in \mathbb{R}^X} ||\mathcal{G}[t, f]||_{\mathcal{M}}. \tag{3.10}$$

The function $|| \cdot ||_{\mathcal{MF}}$ satisfies all properties of the norm with the exception that it is not bounded on $\mathcal{MF}_0$. However, the subset of $\mathcal{MF}_0$ on which it is finite forms a vector space that contains $\mathcal{PF}$. To this end, define

$$\mathcal{MF} \equiv \left\{\mathcal{G} \in \mathcal{MF}_0 : ||\mathcal{G}||_{\mathcal{MF}} < \infty\right\}.$$

Then $\mathcal{MF}$ is a normed vector space with norm $|| \cdot ||_{\mathcal{MF}}$.

Theorem 3.4.1. *$\mathcal{MF}$ is a normed vector subspace of $\mathcal{MF}_0$ under $|| \cdot ||_{\mathcal{MF}}$.*

Proof. The vector space structure of $\mathcal{MF}_0$ has already been discussed. To see that $|| \cdot ||_{\mathcal{MF}}$ is a norm, note that for $\mathcal{G} \in \mathcal{MF}$, $\alpha \in \mathbb{R}$,

$$||\alpha\mathcal{G}||_{\mathcal{MF}} = \sup_{t,f} ||\alpha\mathcal{G}||_{\mathcal{M}} = |\alpha|\, ||\mathcal{G}||_{\mathcal{MF}}.$$

Additionally, if $\mathcal{G} \neq 0$, then there must exist some t, f such that $||\mathcal{G}\left[t, f\right]|| > 0$ and so $||\mathcal{G}||_{\mathcal{MF}} > 0$ as well. For the triangle inequality,

$$\begin{aligned}
||\mathcal{G}_1 + \mathcal{G}_2||_{\mathcal{MF}} &= \sup_{t \in \mathcal{T}, f \in \mathbb{R}^X} ||\mathcal{G}_1\left[t, f\right] + \mathcal{G}_2\left[t, f\right]||_{\mathcal{M}} && (3.11) \\
&\leq \sup_{t \in \mathcal{T}, f \in \mathbb{R}^X} ||\mathcal{G}_1\left[t, f\right]||_{\mathcal{M}} + ||\mathcal{G}_2\left[t, f\right]||_{\mathcal{M}} && (3.12) \\
&\leq \sup_{t \in \mathcal{T}, f \in \mathbb{R}^X} ||\mathcal{G}_1\left[t, f\right]||_{\mathcal{M}} + \sup_{t \in \mathcal{T}, f \in \mathbb{R}^X} ||\mathcal{G}_2\left[t, f\right]||_{\mathcal{M}} && (3.13) \\
&= ||\mathcal{G}_1||_{\mathcal{MF}} + ||\mathcal{G}_2||_{\mathcal{MF}} < \infty && (3.14)
\end{aligned}$$

So $||\cdot||_{\mathcal{MF}}$ is indeed a norm.

$\mathcal{MF}$ is a vector subspace because it contains the zero vector and is closed under vector addition and scalar multiplication. In particular, $||0||_{\mathcal{MF}} = 0 < \infty$ and closure under linear operations follows from the properties of the norm. $\square$

It may be asked whether $\mathcal{MF}$ is complete and therefore Banach. The answer is no; it is easy to create sequences in $\mathcal{MF}$ with an unbounded norm in the limit. However, this fact will not be particularly restrictive for the purpose of analysis, since the subset of population-based optimizers is a closed subset of $\mathcal{MF}$.

Returning to the goal of this section, the following proposition holds.

Proposition 3.4.2. *$\mathcal{PF}$ is a closed, convex subset of $\mathcal{MF}$.*

Proof. First of all, if $\mathcal{G} \in \mathcal{PF}$ then $||\mathcal{G}||_{\mathcal{MF}} = 1 < \infty$, so $\mathcal{PF} \subseteq \mathcal{MF}$.

To show that $\mathcal{PF}$ is closed, let $||\mathcal{G}_n - \mathcal{G}||_{\mathcal{MF}} \to 0$ for $\{\mathcal{G}_n\} \subseteq \mathcal{PF}$. Then for all t, f, $||\mathcal{G}_n\left[t, f\right] - \mathcal{G}\left[t, f\right]||_{\mathcal{M}} \to 0$, and for all $A \in \mathcal{B}_\tau$,

$$\begin{aligned}
\mathcal{G}\left[t, f\right](X) &= \lim_n \mathcal{G}_n\left[t, f\right](X) = 1, && (3.15) \\
\mathcal{G}\left[t, f\right](A) &= \lim_n \mathcal{G}_n\left[t, f\right](A) \geq 0. && (3.16)
\end{aligned}$$

That is, $\mathcal{G}\left[t, f\right]$ is a probability measure, so $\mathcal{G} \in \mathcal{PF}$.

To establish convexity, let $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{PF}$, and let $\alpha \in [0, 1]$. Set $\mathcal{G} = \alpha\mathcal{G}_1 + (1 - \alpha)\,\mathcal{G}_2$. Then for all t, f, and A,

$$\begin{aligned}
\mathcal{G}\left[t, f\right](X) &= \alpha\mathcal{G}_1\left[t, f\right](X) + (1 - \alpha)\,\mathcal{G}_2\left[t, f\right](X) = 1, && (3.17) \\
\mathcal{G}\left[t, f\right](A) &= \alpha\mathcal{G}_1\left[t, f\right](A) + (1 - \alpha)\,\mathcal{G}_2\left[t, f\right](A) \geq 0, && (3.18)
\end{aligned}$$

and therefore $\mathcal{G} \in \mathcal{PF}$. $\square$

3.4.2 Vector Subspaces of $\mathcal{MF}$

Elements of $\mathcal{MF}$ are "optimizers" only in a loose formal sense. Signed measures cannot be sampled, and thus the majority of objects in $\mathcal{MF}$ do not serve the purpose of optimization. Thus when a distinction is needed, elements of $\mathcal{MF}$ will be termed as *generalized optimizers*. The space $\mathcal{MF}$ is useful because it permits a structural analysis of $\mathcal{PF}$. In order to further this analysis, the properties of $\mathcal{PF}$ can be carried over to $\mathcal{MF}$.

The definitions of trajectory-restrictedness and information-restrictedness carry over verbatim to elements of $\mathcal{MF}$. A generalized optimizer $\mathcal{G}$ is trajectory-restricted if $\mathcal{G}[t, f] = \mathcal{G}[t, g]$ whenever $f(x) = g(x)$ for all $x \in t$. The optimizer $\mathcal{G}$ is information-restricted if there exists an information function I such that $\mathcal{G}[t, f] = \mathcal{G}[t, g]$ whenever $I(f, x) = I(g, x)$ for all $x \in t$. Then let $\mathcal{MF}_{\text{tr}}$ be the class of trajectory-restricted generalized optimizers, and let $\mathcal{MF}_{\text{ir}}$ be the class of information-restricted generalized optimizers. Clearly, $\mathcal{O}_{\text{tr}} \subseteq \mathcal{MF}_{\text{ir}}$ and $\mathcal{O}_{\text{ir}} \subseteq \mathcal{MF}_{\text{ir}}$.

It is more difficult to extend computability to generalized optimizers, since it does not make sense to compute a signed measure. However, an abstract definition may be reached through closure under vector operations, and an abstraction will suffice for this analysis.

Definition 3.4.1. *A generalized optimizer $\mathcal{G} \in \mathcal{MF}$ is computable if any one of the following conditions hold:*

- $\mathcal{G} \in \mathcal{PF}$ *and* $\mathcal{G}$ *is computable, or*
- $\exists \alpha \in \mathbb{R}$ *and* $\exists \mathcal{C} \in \mathcal{MF}$ *with* $\mathcal{C}$ *computable such that* $\mathcal{G} = \alpha\mathcal{C}$*, or*
- $\exists \mathcal{C}_1, \mathcal{C}_2 \in \mathcal{MF}$*, both computable, such that* $\mathcal{G} = \mathcal{C}_1 + \mathcal{C}_2$*.*

Define a generalized optimizer to be polynomially computable if it can be constructed in a similar fashion from linear operations over a base of polynomially computable members of $\mathcal{PF}$. Define $\mathcal{MF}^{\text{co}}$ to contain computable generalized optimizers, and $\mathcal{MF}^{\text{poly}}$ to contain polynomially computable generalized optimizers. The properties of computability, trajectory-restrictedness, and information-restrictedness each define a proper vector subspace of $\mathcal{MF}$.

Proposition 3.4.3. *The following subsets of* $\mathcal{MF}$ *are proper vector subspaces:* $\mathcal{MF}_{\text{ir}}, \mathcal{MF}_{\text{tr}}, \mathcal{MF}^{\text{co}}, \mathcal{MF}^{\text{poly}}$.

Proof. It should be clear that there exist optimizers in $\mathcal{PF}$ that are neither information-restricted, trajectory-restricted, or computable. As an example, consider the omniscient optimizer that outputs a true global optimum at every time step. It is not information- (or trajectory-) restricted, and if it were computable, it could solve the halting problem by optimizing over functions that map each Turing machine to a boolean indicating whether it halts. Thus all of the subsets under consideration are proper.

In addition, the zero vector is trivially information and trajectory-restricted. It is also computable, being the scalar product of 0 with any computable optimizer in $\mathcal{PF}$. By definition, computable and polynomially computable generalized optimizers are closed under vector operations, so $\mathcal{MF}^{\text{co}}$ and $\mathcal{MF}^{\text{poly}}$ are both vector subspaces of $\mathcal{MF}$. Also, as was discussed in Section 3.3.3, $\mathcal{O}_{\text{tr}}$ and $\mathcal{O}_{\text{ir}}$ are both closed under the vector operations, and repeating those arguments from $\mathcal{MF}_{\text{tr}}$ and $\mathcal{MF}_{\text{ir}}$ leads to the conclusion that each of these is a vector subspace of $\mathcal{MF}$. $\square$

In addition, the intersection of any two vector subspaces is a vector subspace, and thus $\mathcal{MF}^{\text{co}}_{\text{ir}} = \mathcal{MF}_{\text{ir}} \bigcap \mathcal{MF}^{\text{co}}$ is a vector subspace. The same is true for other intersections, similarly denoted by $\mathcal{MF}^{\text{co}}_{\text{ir}}$, $\mathcal{MF}^{\text{co}}_{\text{tr}}$, $\mathcal{MF}^{\text{poly}}_{\text{ir}}$, and $\mathcal{MF}^{\text{poly}}_{\text{tr}}$. In light of the discussion in Section 3.3.3, it is then clear that each of the optimizer classes $\mathcal{O}_{\text{ir}}$, $\mathcal{O}_{\text{tr}}$, $\mathcal{O}^{\text{co}}$, and $\mathcal{O}^{\text{poly}}$ as well as their intersections are convex subsets of their respective analogues in $\mathcal{MF}$.

The final question to be answered in this section is whether these convex subsets are closed under the norm $|| \cdot ||_{\mathcal{MF}}$. The answer is yes. For the sake of simplicity, write $\mathcal{MF}^*_*$ so that $*$ varies over co, poly, ir, tr, and their various combinations, and let O^*_* be the relevant convex subset of $\mathcal{MF}^*_*$.

Proposition 3.4.4. $\mathcal{O}^*_*$ *is norm-closed in* $\mathcal{MF}^*_*$ *and* $\mathcal{MF}$.

Proof. The set of probability-valued functions $\mathcal{PF}$ is a closed subset of $\mathcal{MF}$, and $\mathcal{MF}^*_*$ is a vector subspace of $\mathcal{MF}$. thus $\mathcal{O}^*_* = \mathcal{PF} \bigcap \mathcal{MF}^*_*$ is closed in $\mathcal{MF}^*_*$ as a consequence of elementary function analysis (the intersection of a closed set and a vector subspace is closed). $\square$

It is somewhat surprising that this result is so easily obtained, since it is not immediately clear how to prove directly that the limit of computable optimizers is computable, or that the limit of information-restricted optimizers is also information-restricted (a direct proof of norm closure for trajectory-restricted optimizers is easier). However, the result is intuitive. For example, choose two optimizers, $\mathcal{G}_1$ computable and $\mathcal{G}_2$ non-computable. Consider the line between them, parameterized by $\alpha \in [0, 1]$. Notice that $\alpha\mathcal{G}_1 + (1 - \alpha)\mathcal{G}_2$ is non-computable for all choices of α other than one. That is, the boundary between computability and non-computability is sharp, and there are innumerably more non-computable optimizers than there are computable ones.

3.5 Conclusion

The main conclusion from the detailed analysis in this chapter is that stochastic optimizers are a closed and convex subset of a normed vector space. It is worthwhile to consider the implications of this result. Most importantly, optimizers are vectors and between any two optimizers there exists an entire range of optimizers given by the convex combinations of the two optimizers at the endpoints. That is, given $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{O}$ and $\alpha \in (0, 1)$, then $\mathcal{G} \equiv \alpha\mathcal{G}_1 + (1 - \alpha)\,\mathcal{G}_2 \in \mathcal{O}$. Also, $\mathcal{G}$ is easily computable whenever $\mathcal{G}_1$ and $\mathcal{G}_2$ are computable. The optimizer $\mathcal{G}$ is computed by flipping a coin with bias α at each generation to decide whether $\mathcal{G}_1$ or $\mathcal{G}_2$ will be used to generate the next population. The fact that $\mathcal{G}$ is computable does not imply that $\mathcal{G}$ is a good algorithm. And yet, as will be seen in Chapter 8, under certain performance criteria, there may exist fitness functions where $\mathcal{G}$ is better than either $\mathcal{G}_1$ or $\mathcal{G}_2$.

Furthermore, given any two optimizers, it is possible to compute a meaningful distance between the two, $||\mathcal{G}_1 - \mathcal{G}_2||_{\mathcal{MF}}$. In practice, the distance between two optimizers is not nearly so important as the difference in their performance on one or more fitness functions. This topic will be taken up again in Chapter 7, where performance criteria will be used to analyze these performance differences.

The definitions and concepts presented in this chapter form the basis for future exploration of the nature and performance of optimizers. For example, the next chapter shows that this formalism is in fact sufficient to describe all common evolutionary algorithms. Chapter 5 studies which optimizers are continuous using this formalism, which leads to the conclusion in Chapter 7 that the performance of an optimizer on an objective changes continuously with either the optimizer or the objective. Finally, Chapter 9 extends the No Free Lunch theorems to arbitrary measure spaces using the concepts presented here. Thus the explicit formalization of optimizers as mathematical objects makes it possible to prove powerful theorems that aid in assessing the value of practical optimization methods.

Chapter 4

A Unified View of Population-Based Optimizers

In the previous chapter, stochastic optimizers were formalized as functions from a prior trajectory to a probability distribution over the next evaluation points. These optimizers were described as proposing one evaluation point at a time. In evolutionary algorithms, evaluation points are generated in batches for parallel evaluation rather than one point at a time. Such optimizers will be termed *population-based optimizers*. The best-known population-based optimizers are evolutionary algorithms. The goal of this effort is to produce a unified analytic approach to evolutionary computation that relates this field to general methods of iterative optimization.

4.1 Population-Based Optimizers

In this chapter, population-based optimizers will be built up from stochastic optimizers like those in the last chapter. This section lays out the goals and definitions that will guide this process.

4.1.1 Motivation and Goals

From a formal perspective, the introduction of populations to be evaluated in parallel changes the nature of an optimizer in one respect only: a population-based optimizer must be able to generate an entire population without depending on evaluations of earlier members of the same population. Parallelization is an implementation detail that can be applied to any optimizer with appropriate dependencies.

Since the most common population-based approaches are evolutionary algorithms, the terminology used here will be drawn from that field as well. Each batch will be termed a *population*, and successive populations will be referred to as *generations*. A particular evaluation point may be referred to as an *individual*, and the objective function may be called a *fitness* function, all following the lexicon of evolutionary algorithms based on extended Darwinian analogies.

A population-based optimizer is nonetheless a stochastic optimizer and can be identified with an element in $\mathcal{PF}$, usually in $\mathcal{O}^{\text{co}}_{\text{tr}}$. The choice to study $\mathcal{PF}$ rather than starting out with populations was made because it allows for a direct comparison among optimizers with different population sizes as well as comparison with typologically distinct approaches to optimization. It also makes it possible to study optimizers with dynamic population sizes, although this dissertation will not evaluate such methods further. Rather, a population-based optimizer will be assumed to have a fixed population size K; that is, it will generate batches of K evaluation points using the same information.

Many evolutionary algorithms can be built up from modular components using the convolution operator from the last chapter. This process is analogous to traditional analyses using genetic operators. These components can be defined individually, and their modular structure can be useful for developing general theorems. Selection, recombination and mutation will be studied as component classes that can be used to abstractly characterize evolutionary algorithms. This dissertation proposes that an evolutionary algorithm can be identified with the convolution of selection, recombination, and mutation operators.

To solidify the claim, equations will be presented that define the most common genetic algorithms and evolution strategies using this modular approach. Similar analysis will be performed for some quasi-evolutionary methods that will highlight some of the ways in which these methods both conform to and deviate from the standard evolutionary computation model.

4.1.2 Formalities

A population-based optimizer $\mathcal{G}$ with population size $K > 0$ can be represented as a sequence of K separate optimizers $\mathcal{G}_1, \cdots, \mathcal{G}_K \in \mathcal{PF}$ (not necessarily distinct), each of which is used to generate one individual per population. Then a trajectory can be broken up into populations, with one optimizer assigned to each slot in the population.

This choice of representation requires tools and notation to convert between trajectories of evaluation points and population histories. Thus one may write $\mathcal{G}\left[t, f\right] = \mathcal{G}_{k(t)}\left[t, f\right]$, where $k(t) \equiv 1 + (|t| \mod K)$ is the index of the individual in the population currently being constructed. The function $k(t)$ will be used repeatedly below. A population is an element in the product space X^K consisting of K copies of the search space X. A trajectory $t \in \mathcal{T}[X]$ can be broken up into a history of populations $H = h^1, h^2, h^3, \cdots$ with $h^i \in X^K$ using the mapping $h^{i,k} = t^{(i-1)K+k+1}$, recalling that trajectories are indexed with superscripts. Let $H(t)$ be the history of complete populations in the trajectory t, so that $H(t)$ ignores any elements in t with index greater than $\lfloor |t|/K \rfloor$. Then $H(t)$ is a trajectory over populations, i.e. $H(t) \in \mathcal{T}\left[X^K\right]$. To complete the setup, let $\text{traj}(H)$ convert a population history $H \in \mathcal{T}\left[X^K\right]$ to a trajectory in $\mathcal{T}\left[X\right]$ via the mapping $\text{traj}(H)^j = H^{\lfloor j/K \rfloor, 1+(j \mod K)}$. Then $\text{traj}(H(t)) = t$ if and only if the length of t is a multiple of K, i.e. $|t| = K\lfloor |t|/K \rfloor$; otherwise, it truncates the end of t at the last population boundary. The notation $\mathcal{G}[H(t), f]$ will be used to mean $\mathcal{G}[\text{traj}(H(t)), f]$ when this notation is clear from the context.

A population-based optimizer is distinguished by the fact that it respects the population boundary, and new populations can only be generated based on information available from prior populations. That is, to be a population-based optimizer, an optimizer must be able to evaluate points in parallel. This restriction can be represented in terms of an information function as was done for the set $\mathcal{O}_{\text{ir}}$.

Definition 4.1.1. *An optimizer $\mathcal{G} \in \mathcal{PF}$ is a population-based optimizer of population size K if there exists an information function $I : \mathbb{R}^X \times X \to \mathcal{T}[X]$ such that $\mathcal{G}\left[t, f\right] = \mathcal{G}\left[t, g\right]$ whenever $I(f, x) = I(g, x)$ for all $x \in \text{traj}(H(t))$. If $I(f, x) = f(x)$, then $\mathcal{G}$ is also trajectory-restricted.*

This definition might seem excessive at first, since one might imagine it sufficient to require $\mathcal{G}\left[t, f\right] = \mathcal{G}_k(t)\left[H(t), f\right]$ for some $\mathcal{G}_1, \cdots, \mathcal{G}_K$, but such a conceptualization is inaccurate, since populations may be generated from a joint distribution. For example, although an optimizer may not generate the k^{th} member of the population based on the objective evaluation of the $(k-1)^{th}$ member, it may need to inspect the identity of the $(k-1)^{th}$ member, either to avoid duplication (e.g. tabu search), to promote population diversity, or to alter its probability distribution in some other way.

Let $\mathcal{PBO}_K$ be the set of population-based optimizers of size K. Then $\mathcal{PBO}_K \subset \mathcal{O}_{\text{ir}}$. Also, $\mathcal{PBO}_K \subseteq \mathcal{PBO}_{nK}$ for $n \geq 1$. Following the conventions adopted thus far, let $\mathcal{PBO}_{K,\text{tr}}$ be the set of trajectory-restricted population-based optimizers (which includes most evolutionary algorithms). Let $\mathcal{PBO}_K^{\text{co}}$ be the set of computable population-based optimizers of size K, and so on. Notice that $\mathcal{PBO}_K$ and its just-mentioned subsets extend naturally to $\mathcal{MF}$ via closure under vector operations and that these extensions form proper vector subspaces of $\mathcal{MF}$. Thus once again $\mathcal{PBO}_K$ and its subsets form closed, convex sets inside of vector subspaces of $\mathcal{MF}$.

4.1.3 Constructive Operators

Evolutionary algorithms will be built up through constructive operations and analyzed through the building blocks of these operations. In this section, a population-based optimizer $\mathcal{G} \in \mathcal{PBO}_K$ is associated with K optimizers $\mathcal{G}_1, \ldots, \mathcal{G}_K$, with $\mathcal{G}[t, f] = \mathcal{G}_{k(t)}[t, f]$, where $k(t)$ is the population indexing function from the previous section. One generation of $\mathcal{G}$ samples each of the $\mathcal{G}_k$ in turn.

Expanding based on the definitions, the probability density of a particular population $P \in X^K$ given a prior trajectory t and an objective f is

$$\mathbb{P}_{\mathcal{G}}\left(dP \mid t, f\right) = \prod_{k=1}^{K} \mathcal{G}_k \left[t \cup \left(\bigcup_{j=1}^{k-1} P_j \right), f \right] (dP_k) . \tag{4.1}$$

Notice that $\mathcal{G}[t, f](dx)$ is a conditional probability over trajectories and/or fitness functions and can be written as $\mathcal{G}[t, f](dx) = \mathbb{P}_{\mathcal{G}}\left(dx \mid t, f\right)$ with its usual meaning. The concatenation over P_j in Equation 4.1 reflects the fact that the population is sampled jointly, and if Equation 4.1 is rewritten as

$$\mathbb{P}_{\mathcal{G}}\left(dP \mid t, f\right) = \prod_{k=1}^{K} \mathbb{P}_{\mathcal{G}_k} \left(dP_k \mid t \cup \left(\bigcup_{j=1}^{k-1} P_j \right), f \right) , \tag{4.2}$$

then it is clear that Equation 4.1 is just an application of Bayes' rule to the probability of the population.

Quite often, evolutionary algorithms generate each individual of the next population independently of the others. In this case, the joint distribution over individuals in the population factorizes, and such an optimizer is termed *factorial*. In some evolutionary algorithms, population members are not only independent but also identically distributed. Such algorithms are termed *factorially homogeneous*, and the distribution from which each individual is drawn is termed the *factorial base* of the optimizer.

Definition 4.1.2. *An optimizer* $\mathcal{G} \in \mathcal{PBO}_K$ *is factorial if there exist* $\mathcal{G}_1, \dots, \mathcal{G}_K \in \mathcal{PF}$ *such that* $\mathbb{P}_{\mathcal{G}}\left(dP \mid t, f\right) = \prod_{k=1}^{K} \mathcal{G}_k\left[t, f\right]\left(dP_k\right)$

Definition 4.1.3. *An optimizer* $\mathcal{G} \in \mathcal{PBO}_K$ *is homogeneous if there exists* $\mathcal{G}' \in \mathcal{PF}$ *such that* $\mathbb{P}_{\mathcal{G}}\left(dP \mid t, f\right) = \prod_{k=1}^{K} \mathcal{G}'\left[t \cup \left(\bigcup_{j=1}^{k-1} P_j\right), f\right]\left(dP_k\right)$*, and* $\mathcal{G}'$ *is the base of* $\mathcal{G}$.

Definition 4.1.4. *An optimizer* $\mathcal{G} \in \mathcal{PBO}_K$ *is factorially homogeneous if there exists* $\mathcal{G}' \in \mathcal{PF}$ *such that* $\mathbb{P}_{\mathcal{G}}\left(dP \mid t, f\right) = \prod_{k=1}^{K} \mathcal{G}'\left[t, f\right]\left(dP_k\right)$*, and* $\mathcal{G}'$ *is the factorial base of* $\mathcal{G}$.

The following proposition is then sufficient to construct factorial and factorially homogeneous population-based algorithms using the trajectory-truncation operator. Its proof follows immediately from the definitions above.

Proposition 4.1.1. *An optimizer* $\mathcal{G} \in \mathcal{PBO}_K$ *is factorial if and only if there exist* $\mathcal{G}_1, \dots, \mathcal{G}_K \in \mathcal{PF}$ *such that* $\mathcal{G}[t, f] = \triangleleft_{k(t)}\mathcal{G}_{k(t)}[t, f] = \mathcal{G}_{k(t)}[H(t), f]$*, and factorially homogeneous if and only if* $\mathcal{G}[t, f] = \triangleleft_{k(t)}\mathcal{G}_1[t, f] = \mathcal{G}_1[H(t), f]$.

In sum, given a suitable set of optimizers $\mathcal{G} \subseteq \mathcal{O}_{\mathrm{ir}}$, a non-factorial population-based optimizer can be written as $\mathcal{G} = \mathcal{G}_{k(t)}$, a factorial one as $\mathcal{G} = \triangleleft_{k(t)}\mathcal{G}_{k(t)}$, and a factorially homogeneous one as $\mathcal{G} = \triangleleft_{k(t)}\mathcal{G}_1$. This technique will be used to construct common evolutionary algorithms in the next section.

4.1.4 Examples: Random Walkers

A simple example of a population-based optimizer should help to make these concepts concrete. Consider the case of K parallel biased random walkers in the search space $\mathbb{Z}$. Given a starting point x, a biased random walker flips a weighted coin to decide either to advance one step to $x+1$ with probability p or to go backwards one step to $x-1$ with probability $1-p$. Thus a single random walker can be represented for $|t|>0$ as

$$\mathcal{RW}{<}p,x{>}[t,f](Z_{|t|+1}) = \begin{cases} 1-p & \text{if } Z_{|t|+1} = t_{-1}+1 \\ p & \text{if } Z_{|t|+1} = t_{-1}-1 \\ 0 & \quad otherwise, \end{cases} \tag{4.3}$$

with $\mathcal{RW} < p,x > [\emptyset, f](Z_1) = \delta_x\left(Z_1\right)$ to start at x. The variables $< p, x >$ indicate that this biased random walker is parameterized by the weight of the coin and the starting point. Parameterized optimizers will be used extensively from this point. Since the random walker is easily computable and never looks at the objective evaluation, it is clear that $\mathcal{RW} \in \mathcal{PBO}_1^{\text{co}}$. A population of K random walkers can be constructed in several ways. Because such a population never evaluates the objective function, it is trivial that any such population is contained in $\mathcal{PBO}_K^{\text{co}}$.

Factorial and Homogeneous Independent Paths. Choose K random walkers, each with identical bias p and starting at zero. Let each walker ignore every other walker, so that the K walkers follow K independent paths. This optimizer is given by

$$\mathcal{PRW}{<}p{>}[t,f] = \triangleleft_K \mathcal{RW}{<}p,0{>}[t,f]. \tag{4.4}$$

Each walker rewinds the path t to its last decision t_{-K} using trajectory truncation, and then continues its own path. This optimizer is homogeneous, because each individual in the population is sampled from the same base optimizer ($\triangleleft_K \mathcal{RW}$). However, $\mathcal{PRW}$ is not factorially homogeneous because $\triangleleft_i \mathcal{RW}{<}p,0{>} \neq \triangleleft_j \mathcal{RW}{<}p,0{>}$ unless $i=j$.

Homogeneous Dependent Paths. Choose K random walkers, each with bias p and starting at zero. Rather than rewinding paths, each walker builds on the path determined by the previous walker. This optimizer is in $\mathcal{PBO}_K$. It is homogeneous, but it is uninteresting because it is equivalent to $\mathcal{RW}{<}p,0{>}$.

Dependent Paths. Choose K random walkers, each with a different bias p_i, each starting at zero. Again, each walker builds on the path determined by the previous walker, but in this case the walker uses separate weighted coins for different members of the population.

$$\mathcal{PRW}\!<\!p_1,\ldots,p_K\!>\![t,f] = \mathcal{RW}\!<\!p_{k(t)},0\!>\![t,f]. \tag{4.5}$$

Thus this optimizer maintains a single random path through the search space but uses K different weighted coins in a cycle. This optimizer is neither factorial nor homogeneous.

Independent Paths. Choose K random walkers, each with a different bias p_i, each starting at zero. Let each walker ignore every other walker, so that the K walkers follow K independent paths. This optimizer is given by

$$\mathcal{PRW}\!<\!p_1,\ldots,p_K\!>\![t,f] = \triangleleft_K \mathcal{RW}\!<\!p_{k(t)},0\!>\![t,f]. \tag{4.6}$$

Each walker rewinds the path t to its last decision t_{-K} using trajectory truncation, and then continues its own path. The walkers are independent and non-homogeneous, since they use different biases. Because the walkers are independent of each other, this optimizer is factorial.

4.1.5 Example: Simulated Annealing

The four population-based optimizers in the last section illustrate simple construction procedures. However, they are weak as optimizers because they do not take objective evaluations into account. This section develops simulated annealing as an example that does utilize objective evaluations. This example is informative because it contains meaningful substructure and is a well-known and commonly used optimizer in its own right.

Recall that simulated annealing consists of a sequence of objective evaluations. At each time step, there is an accepted solution x. At each time step, a new solution y is proposed. The objective value $f(y)$ is computed, and y replaces x as the accepted solution with probability

$$A(y,x,f,T) = \exp\left(\frac{1}{T}\left[f(x) - f(y)\right]\right) \wedge 1, \tag{4.7}$$

where the infix operator $\wedge$ indicates the minimum of its arguments, so that y is always accepted if $f(y) < f(x)$. Simulated annealing can thus be seen as a population-based algorithm of size two, with each population consisting of the accepted and proposed solutions.

In addition, simulated annealing requires a proposal distribution that is used to generate y from the accepted solution x. The proposal distribution depends on the search space. For this example, let $X = \mathbb{R}^d$, i.e. a d-dimensional Euclidean space. Then a suitable proposal distribution is a the multivariate Gaussian distribution, $\mathcal{N}\left<\mu, \Sigma\right>$. Suppose for this example that the covariance matrix is fixed to the identity, although most instances of simulated annealing dynamically alter the covariance matrix to keep the acceptance probability close to 0.23. Let the trajectory t track the accepted solution and the proposed solution in alternation, so that each point t^n in the trajectory is the accepted solution at the n^{th} time step if n is odd, and the proposed solution if n is even. Then set $\mu = t^{-1}$, the last accepted solution in the trajectory. Then the proposal distribution is given by

$$\mathcal{P}[t, f] = \mathcal{N}{<}t^{-1}, I{>} \, . \tag{4.8}$$

Given a proposed y and an accepted solution x, simulated annealing performs a Bernoulli trial to determine whether to accept y or keep x. Let $\mathcal{B}\left<p, y, x\right>$ be a Bernoulli distribution that produces y with probability p and x with probability $1 - p$. Then the acceptance step for simulated annealing is an optimizer given by

$$\mathcal{A}[t, f] = \mathcal{B}\big\langle A(t^{-1}, t^{-2}, f, T(|t|/2)), t^{-1}, t^{-2}\big\rangle \, , \tag{4.9}$$

recalling that t^{-1} contains the proposal and t^{-2} the accepted solution. The temperature $T(n)$ is assumed to be a function of the length of the trajectory, commonly $T(n) = 1/\log n$.

Simulated annealing can thus be viewed as a population-based optimizer $\mathcal{SA}$ of size 2 with $\mathcal{SA}_1 = \mathcal{A}$ and $\mathcal{SA}_2 = \mathcal{P}$. The starting population (t_1, t_2) is initialized randomly, and thenceforth $\mathcal{SA}_1$ and $\mathcal{SA}_2$ are used in alternation to accept and propose solutions.

The optimizer $\mathcal{SA}$ is neither factorial nor homogeneous, but it is information-restricted. As defined in the prior paragraph, $\mathcal{SA} \in \mathcal{PBO}_2^{\text{co}}$, since only $\mathcal{SA}_1 = \mathcal{A}$

depends on the objective evaluations and $\mathcal{SA}_2 = \mathcal{P}$ does not. The order of these two steps is important to this construction, because if the proposal step was performed first $\mathcal{SA}$ would violate the definition of $\mathcal{PBO}_2$ by relying on the objective evaluation of the first member of the population to generate the second before the population was completely constructed. While this distinction seems arbitrary in the case of simulated annealing, it captures the difference between a batch optimizer and a one-step-at-a-time optimizer. The distinction is computationally relevant, because a population-based optimizer meeting the definition of $\mathcal{PBO}_K$ can compute any necessary function evaluations in K parallel processes, whereas the same is not true for all members of $\mathcal{O}_{\text{tr}}$.

However, this arbitrariness captures a relevant fact about simulated annealing, namely, that it does not match well with our natural intuitions about a population-based algorithm. In fact, any evaluation of the performance of simulated annealing would not change if the optimizer had been defined as $\mathcal{SA} = \mathcal{A} \star \mathcal{P}$. In this case, $\mathcal{SA} \in \mathcal{O}^{\text{co}}_{\text{tr}} = \mathcal{PBO}^{\text{co}}_{1,\text{tr}}$, a computable trajectory-restricted optimizer. By contrast, $\mathcal{P} \star \mathcal{A} \notin \mathcal{O}^{\text{co}}_{\text{tr}}$, even though it generates almost identical trajectories as $\mathcal{A} \star \mathcal{P}$, because it must evaluate the objective during sampling. The fact that $\mathcal{A} \star \mathcal{P} \in \mathcal{O}^{\text{co}}_{\text{tr}}$ but $\mathcal{P} \star \mathcal{A} \notin \mathcal{O}^{\text{co}}_{\text{tr}}$ is simply a quirk of the chosen formalism. A formalism that resolves this oddity would be more complex to describe and analyze. In such a formalism, one could abandon information-restrictedness in favor of a treatment based solely on computability, but analytically, it is easier to work with information-restrictedness than computability.

The profusion of variables, operators, and symbols in this example may seem unnecessary at first. After all, it is possible to write pseudocode for simulated annealing with less effort than it took to describe $\mathcal{SA}$. However, the formalism makes it possible to compare simulated annealing directly with other optimization routines in a way that pseudocode does not allow. For instance, the classic evolutionary strategy known as the $(1+1)$–ES is the norm-limit of $\mathcal{SA}$ as the temperature goes to zero, as is shown in Theorem 4.2.3.

In addition, the example of simulated annealing has made use of several components and techniques that will be used in defining evolutionary algorithms. The proposal distribution $\mathcal{P}$ plays the role of a mutation operator in evolutionary methods, randomly altering a previously evaluated point.

The acceptance optimizer $\mathcal{A}$ mirrors the role of selection in evolutionary methods. The convolution $\mathcal{A} \star \mathcal{P}$ is analogous to the exact form of an evolutionary algorithm, combining selection and variation in sequence. In fact, by the formal definitions that will be given in the next section, simulated annealing is an evolutionary algorithm. Conversely, one might say that evolutionary algorithms are stochastic Monte Carlo optimization routines. It makes no difference which category subsumes the other. The fact is that there is no formal difference between Monte Carlo optimization and evolutionary optimization, something that only becomes clear when evolutionary algorithms are formally analyzed. [1] With this goal in mind, the discussion now turns explicitly to a formalization of evolutionary algorithms.

4.2 Evolutionary Algorithms

In this section, the most common evolutionary algorithms are represented in the formal framework of the previous section. This process demonstrates that the formalization in this chapter and the preceding one do apply to complex practical algorithms. Also, these definitions will be used repeatedly in Chapters 5 and 7 to prove that the performance of most evolutionary algorithms is continuous as the fitness function changes.

4.2.1 Characteristics of an Evolutionary Algorithm

The core characteristics of an evolutionary algorithm are based on an analogy with Darwinian principles and include competition within a population, preferential selection of competitive individuals, reproduction among selected individuals, and random variation of selected individuals. These four processes can be realized into evaluation, selection, recombination, and mutation phases. Selection and recombination occur at the level of populations. Variation occurs at the level of the individual. In formal terms, an evolu-

---

[1] In existing literature evolutionary computation is occasionally referred to as a form of Monte Carlo optimization, but this statement is intuitively rather than formally derived. The conclusion follows by formalizing evolutionary algorithms mathematically, because Monte Carlo algorithms have always been described mathematically.

tionary algorithm can be identified as a convolution of three components, one each for selection, recombination, and mutation processes. Evaluation of the fitness function precedes selection. Recombination may be vacuous (asexual reproduction), in which case the algorithm is represented by a convolution of selection and mutation, much as simulated annealing was defined in the last section ($\mathcal{SA} = \mathcal{A} \star \mathcal{P}$).

Each of the phases of an evolutionary algorithm can be described as an optimizer, just as the acceptance phase and proposal phase of simulated annealing were separated out into two different components. Thus the first step in formalizing evolutionary algorithms is to define what principles make an optimizer work as a selection rule, a recombination operator, or a mutation operator. Viewed independently, the optimizers representing each phase are not effective optimizers by themselves in the general case. A selection rule alone is totally ineffective, since it cannot propose new solutions beyond what has already been evaluated. Mutation operators implement a blind random search. Recombination reconfigures evaluated points. In small, discrete spaces, selection plus recombination can be very effective at exploring the space given a sufficiently diverse initial population, but in large spaces, substantial mutation is required to fully explore the space.

4.2.2 Selection, Recombination, and Mutation

An evolutionary algorithm will be defined as the convolution of selection, recombination and mutation. These three phases may be thought of as intermediate steps, each of which creates a full population and hands it off to the next phase. So selection chooses K points from among the previously observed points. Recombination invokes one or more additional selection rules to tack on extra parents and then merges these parents with a crossover rule; this merged output of K individuals is then handed off to the mutation operator, which alters each individual independently. These three stages will now be discussed rigorously one at a time.

Selection in evolutionary algorithms is a filtering task, characterized as follows: Given a set of previously observed individuals, select a group of K individuals to form the basis of the next population. Therefore, the selection process must place zero weight on previously unobserved individuals. Only members of the population history can be selected. Given a trajectory $t \in \mathcal{T}$, define the previously observed individuals in t as $P(t) = \{x \in X : \exists n \text{ s.t.}\, x = t^n\}$. Taking populations into account, a *selection rule* is an optimizer that places zero probability mass on any proposed population that would expand $P(t)$.

Definition 4.2.1. *An optimizer* $\mathcal{S} \in \mathcal{PBO}_K$ *is a selection rule if* $\mathcal{S}\left[t, f\right](A) = 0$ *whenever* $\exists x \in A$ *s.t.* $P(t \cup x) \neq P(t)$*.*

It may seem strange to a practitioner of evolutionary algorithms that the selection rule is allowed to select any member of $P(t)$ and not just the members of the last population $(P(H(t)^{-1}))$. But there are a number of evolutionary methods that select members of populations prior to the last population, such as elitist selection. Methods that store the locally best individual (such as evolution strategies) also need the flexibility to select from previous generations. Furthermore, several recently proposed techniques such as novelty search [125], curiosity search [177], and the evolutionary annealing method proposed in Chapter 11 store members from each population in an archive, making them available for selection.

Recombination combines some number of selected individuals as parents to form a hybrid child. Although traditional recombination methods in genetic algorithms utilize only two parents, other methods use an arbitrary number of parents. In evolution strategies, for example, intermediate crossover averages components across several solutions. A recombination operator first selects the parents for each member of the population and then invokes a crossover rule to combine the parents. The number of selected parents (usually just two) is said to be the *order* of the crossover rule and the recombination operator. Parent selection for an n^{th} order operator stacks n populations on top of the current trajectory. A crossover rule consumes these n populations and leaves a single merged population in their place.

The key feature of a crossover rule is that it should combine only the selected parents. It should therefore be independent of all other components of the input trajectory. It should also ignore the objective value of the selected parents, deferring such judgments to the selection operators. From the perspective adopted in this dissertation, for the k^{th} member of the population the selected parents in a crossover rule of order n are just the k^{th} members of the previous n populations in the trajectory. Define the trajectory $\text{parents}(t, n, k, K) \equiv \bigcup_{i=1}^{n} H(t)^{-i,k}$, recalling that $H(t)$ is the population history of t, negative indices count backwards from the end of the history, and the double index chooses the k^{th} member of the $-i^{th}$ population. Then $\text{parents}(t, n, k, K)$ is the reverse ordered list of the parents available to the crossover rule.

Definition 4.2.2. *An objective-agnostic optimizer $\mathcal{C} \in \mathcal{PBO}_K$ is a crossover rule of order n if there exist $\mathcal{C}_1, \ldots, \mathcal{C}_K \in \mathcal{O}_{\text{ir}}$ such that $\mathcal{C}[t, f] = \mathcal{C}_{k(t)}[t, f]$ and for all $k = 1, \ldots, K$, $t_1, t_2 \in \mathcal{T}$, $\mathcal{C}_k[t_1, f] = \mathcal{C}_k[t_2, f]$ whenever $\text{parents}(t_1, n, k, K) = \text{parents}(t_2, n, k, K)$. That is, a crossover rule is independent of all but the selected parents.*

Such a crossover rule is factorial as defined. It would be possible to define crossover rules to be non-factorial, so that later crossovers depend on the results of earlier ones, but it does not seem necessary. As it is, this definition of crossover accepts a wide range of instantiations that do not necessarily match the concept of crossover in a traditional genetic algorithm. This intuition will be restored with the introduction of crossover masks in Section 4.2.3. With crossover rules defined, the definition of a recombination operator can now be given.

Definition 4.2.3. *An optimizer $\mathcal{R} \in \mathcal{PBO}_K$ is a recombination operator of order n if there exists a sequence of $n-1$ selection rules $\mathcal{S}_1, \ldots, \mathcal{S}_{n-1} \in \mathcal{PBO}_K$ and a crossover rule $\mathcal{C} \in \mathcal{PBO}_K$ of order n such that*

$$\mathcal{R} = \triangleleft \mathcal{S}_1 \star (\triangleleft_2 \mathcal{S}_2 \star (\cdots \star (\triangleleft_{n-1} \mathcal{S}_{n-1} \star \mathcal{C}))) .$$

Operationally, each of the selection rules $\mathcal{S}_i$ are selected in order, with the previous selection hidden by the trajectory-truncation operator. Finally, the crossover rule is invoked to combine the selected points, including the first point selected by an initial selection rule outside of the recombination operator. [2] The convolution is performed with right association so that the results of selection are stacked together and not consumed until the crossover rule is reached. Note that there is only one possible recombination operator of order 1, and it vacuously reproduces the selected population, representing asexual reproduction.

Mutation in evolutionary algorithms alters a single member of a proposed population. Thus a mutation operator is factorial, altering each member of a proposed population independently. Mutation must also be objective-agnostic; it cannot be aware of the fitness of the point it is mutating. In addition, a mutation operator can only vary the individual member of the population that has been proposed to it. That is, a mutation operator must ignore every member of the trajectory except the one that is being mutated. Conversely, a mutation operator cannot simply ignore the individual it is mutating, and so a condition must be included stating that the mutation operator must depend on the object being mutated for at least some trajectories.

Definition 4.2.4. *An optimizer* $\mathcal{V} \in \mathcal{PBO}_K$ *is a mutation operator if* $\mathcal{V}$ *is factorial and objective-agnostic and for all* $1 \leq i \leq K$*, the following two conditions hold:*

- $\forall t_1, t_2 \in \mathcal{T}$, $\mathcal{V}_i\left[t_1, f\right] = \mathcal{V}_i\left[t_2, f\right]$ *whenever* $H(t_1)^{-1,i} = H(t_2)^{-1,i}$, *and*
- $\exists t_1, t_2 \in \mathcal{T}$ *s.t.* $H(t_1)^{-1,i} \neq H(t_2)^{-1,i}$ *and* $\mathcal{V}_i\left[t_1, f\right] \neq \mathcal{V}_i\left[t_2, f\right]$.

A quasi-evolutionary algorithm will be defined as the convolution of a selection rule, a recombination operator and a mutation operator. Recall that the recombination operator contains one or more selection rules and a crossover rule. The recombination operator may also be of order one, in which case it simply copies the initial selection rule. A working definition for a strict evolutionary algorithm will be defined based on crossover masks in Section 4.2.3.

---

[2] The initial selection rule could have been pushed inside the recombination operator, but keeping it outside makes the formal definition of an evolutionary algorithm appear more natural

Definition 4.2.5. *An optimizer* $\mathcal{E} \in \mathcal{PBO}_K$ *is a quasi-evolutionary algorithm if it is not objective-agnostic and if there exist a selection rule* $\mathcal{S}$*, a recombination operator* $\mathcal{R}$ *of order* 1 *or greater, and a mutation operator* $\mathcal{V}$ *such that* $\mathcal{E} = \mathcal{S} \star \mathcal{R} \star \mathcal{V}$*.*

Proposition 4.2.1. *By implication,* $\mathcal{E} \in \mathcal{PBO}_K$ *is also a quasi-evolutionary algorithm if it is not objective-agnostic and there is a selection rule* $\mathcal{S}$ *and a mutation operator* $\mathcal{V}$ *such that* $\mathcal{E} = \mathcal{S} \star \mathcal{V}$*, in which case* $\mathcal{E}$ *has a recombination operator of order* 1*.*

Intuitively, a quasi-evolutionary algorithm first samples one or more selection rules to propose a new parent population consisting of the selected individuals, then recombines the parent population to form a new child population, and finally samples a mutation operator to alter the selected individuals. Because crossover and mutation were defined to be objective-agnostic, it follows from Proposition 3.3.1 that a quasi-evolutionary algorithm is information- or trajectory-restricted if and only if all of its selections are.

The definition of a quasi-evolutionary algorithm and its parts were chosen to exclude algorithms that do not match standard intuitions of how an evolutionary algorithm works. These definitions are restrictive, primarily because the crossover rule and the mutation operator must be objective-agnostic. Otherwise, any population-Markov[3] optimizer $\mathcal{M}$ that is not objective-agnostic would be a quasi-evolutionary algorithm, since $\mathcal{M}$ could be used as a K^{th} order crossover rule, with K selection rules each of which simply pass along one member of the prior population and a vacuous mutation operator that does nothing. The definitions above preclude this possibility.

Now that a formal definition of a quasi-evolutionary algorithm and its components has been given, it is possible to state explicit formulae for common genetic algorithms based on the selection, crossover, and mutation methods that they use. Additionally, further definitions will be proposed that will be used to develop a formal definition a traditional evolutionary algorithm.

---

[3] Defined in the next section.

4.2.3 Genetic Algorithms

Modern genetic algorithms mix and match a variety of selection, crossover, and mutation components to form an optimization routine. This section will review the most common among these components, along with the way in which they are assembled.

With rare exceptions, selection in genetic algorithms is typically restricted to the members of the last population, so that a genetic algorithm unfolds as a sequence of populations, with each population generated directly from the prior population. An optimizer $\mathcal{G} \in \mathcal{PBO}_K$ will be termed *population-Markov* if it depends only on the last population, that is, if $\mathcal{G}[t_1, f] = \mathcal{G}[t_2, f]$ whenever $H(t_1)^{-1} = H(t_2)^{-1}$.

Proposition 4.2.2. *An evolutionary algorithm is population-Markov if and only if its recombination operator and selection rule are, and a recombination operator is population-Markov if and only if each of its subordinate selection rules is.*

Genetic algorithms are population-Markov in general. The most common selection rules historically are proportional selection, tournament selection, and ranking selection.

In *proportional selection*, members of the prior population are selected independently proportional to their fitness in the previous population. Ordinarily, the fitness function is assumed to be positive, and genetic algorithm is maximizing the fitness and so prefers larger fitness values. To use proportional selection for minimization, a function $g > 0$ is introduced so that $g(t, f(\cdot))$ is intended to be positive and increasing as f becomes more optimal. This function g will be called the *modulating function* of proportional selection. If it is desired to maximize f and $f > 0$, then $g(t, x) = |x|$ will prefer the minimal values of $-f$. Proportional selection with this choice of modulating function will be termed *standard proportional selection* or *roulette wheel selection*. A more neutral choice is $g(t, x) = \exp(-x)$; the similarity of this choice with the acceptance probability for simulated annealing should not be missed, and will return in Chapter 11. Given a modulating function g, proportional selection is given explicitly by

$$\mathcal{PS}\left<g\right>[t, f](\{y\}) \propto N_{H(t)^{-1}}(y)\left[g(t, f(y))\right], \tag{4.10}$$

where $N_P\left(y\right)$ is the number of times the individual y appears in the population P. Then N_P is nonzero for at most K points, so the normalization can be computed by summing over the prior population $H(t)^{-1}$.

Proportional selection is highly sensitive to the magnitude of variation in the fitness function and so can become trapped in steep local minima. *Tournament selection* chooses members of the prior population according to their rank in the population in order to maintain diversity within the population. Like proportional selection, tournament selection is factorial and so chooses each member of the prior population based on the same distribution. This distribution selects the best member of the last population with probability q. If the best member is not selected, the second best member is chosen with probability q, and then the third, and the fourth, and so on. If the population is exhausted, the selection wraps back around to the best individual. The parameter q is referred to as the *selection pressure* since high values of q force selection to predominately favor the best individuals in the population. Tournament selection is given explicitly by

$$\mathcal{TS}\left\langle q\right\rangle\left[t,f\right]\left(\left\{y\right\}\right) \propto \left(1-q\right)^{R\left(y,f,H(t)^{-1}\right)}, \tag{4.11}$$

where $R\left(y,f,P\right) \in \mathbb{N}\cup\{\infty\}$ is the rank of the individual y in the population P under the fitness function f, with 0 being the best rank, and $R\left(y,f,P\right) = \infty$ if y does not appear in P, so that the probability of such y being selected is zero. In case of ties, assume later members of the population are ranked higher. Again, $\mathcal{TS}$ is nonzero for at most K points so that the normalization is easily computed.

Like tournament selection, *ranking selection* chooses individuals according to their rank in the prior population, but does so using proportional selection over the rank.Define

$$r_q^{t,f}(x) = \begin{cases} 2-q+2\left(q-1\right)\frac{K-1-R\left(x,f,H(t)^{-1}\right)}{K-1} & \text{if } x \in H(t)^{-1} \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

Then *linear ranking selection* is given by

$$\mathcal{RS}\left\langle q\right\rangle\left[t,f\right]\left(\left\{y\right\}\right) \propto r_q^{t,f}(x), \tag{4.13}$$

where $q \in [1,2]$ is the selection pressure. Notice the similarity to proportional selection. Ranking selection is proportional selection in which the fitness has been replaced with the rank in the population. *Non-linear ranking selection* can be represented in a similar fashion but with more complex detail.

One final aspect of selection in genetic algorithms is *elitism*. Elitism protects the best evaluation point so far from being removed from the population. Elitism can be beneficial to a genetic algorithm because it prevents the algorithm from forgetting the best individual. Given an objective function f, let $best(P,f) \in X$ be the point in the population P with most optimal fitness on f. Then elitist selection alters an entire genetic algorithm $\mathcal{GA} \in \mathcal{PBO}_K$ by preserving $\text{best}(H(t)^{-1})$ as the first member of the population, so that

$$\mathcal{E}\left\langle \mathcal{GA} \right\rangle [t,f] = \begin{cases} \delta_{\text{best}(H(t)^{-1})} & \text{if}\, k(t) = 1 \\ \mathcal{GA}[t,f] & \text{otherwise,} \end{cases} \tag{4.14}$$

where δ_x is the Dirac delta here and below. Importantly, elitism is not a selection rule when defined this way, since it not only selects the best individual, but preserves it from alteration as well.

The distinguishing characteristic of a genetic algorithm is undoubtedly recombination with two parents (sexual reproduction). Standard crossover rules of order 2 include one point crossover, multipoint crossover, and uniform crossover. Most often, the same selection rule is often used to select both parents. Sometimes a strongly selective rule is used to choose the "father" while a more uniform selection rule is used to select the "mother". Either way, the "child" is created to combine properties from the father and the mother.

Because crossover rules are specific to the search space, examples will only be given for the case in which the search space X is a d-dimensional vector space, $X = Y^d$, such as $X = \mathbb{R}^d$ (Euclidean space) or $X = \{0,1\}^d$ (binary space). In this case, many second-order crossover rules can be determined by a random binary vector $M \in \{0,1\}^d$ which will be termed the *crossover mask*. If $M_i = 1$, then the child copies the i^{th} attribute of the father. If $M_i = 0$, then the child copies the i^{th} attribute of the mother. Denote by $\mathbf{1}$ the vector in $\{0,1\}^d$ whose entries are all one, and let $x \otimes y$ be the vector that is the componentwise product of vectors x and y. For a trajectory t, let $p(t)$ be the selected father and $m(t)$ the selected mother, so that $p(t) =$

parents$(t, 2, k(t), K)^{-1}$ and $m(t) = \text{parents}(t, 2, k(t), K)^{-2}$. Define a random variable C_t by

$$C_t = M \otimes p(t) + (\mathbf{1} - M) \otimes m(t). \tag{4.15}$$

Then given a distribution $\mathbb{P}_M$ over M, a *masked crossover rule* is just the distribution of C_t and can be written as

$$\mathcal{C} \left\langle \mathbb{P}_M \right\rangle [t, f](A) = \sum_{z \in \{0,1\}^d} \mathbb{P} \left(C_t \in A \mid M = z \right) \, \mathbb{P}_M \left(z \right), \tag{4.16}$$

Single point, multipoint, and uniform crossover can be defined by specifying $\mathbb{P}_M$. For uniform crossover, the choice of mask is uniformly random,

$$\mathcal{UC}[t, f] = \mathcal{C} \left\langle \text{Uniform} \left(\{0, 1\}^d \right) \right\rangle. \tag{4.17}$$

For single point crossover, a random index $i \in \{1, \ldots, d\}$ is chosen, and the mask is set so that $M_j = 1$ for $j \leq i$ and $M_j = 0$ for $j > i$. In multipoint crossover, a fixed number of random indices $i_1, \ldots, i_n$ are chosen and then sorted. M then alternates between series of zeros and a series of ones, starting with ones and with switches occurring at each of the i_j. Without stating further details, let $\mathcal{SC}$ denote single-point crossover and let $\mathcal{MC}$ represent multipoint crossover.

Masked crossover best captures the characteristic of a traditional genetic algorithm, and an evolutionary algorithm will be defined as a quasi-evolutionary algorithm with a masked crossover rule. A genetic algorithm will be identified as an evolutionary algorithm that is also population-Markov.

Definition 4.2.6. *An optimizer $\mathcal{G} \in \mathcal{PBO}_K$ is an evolutionary algorithm if it is a quasi-evolutionary algorithm with a masked crossover rule. Additionally, $\mathcal{G}$ is a genetic algorithm if it is also population-Markov.*

This definition encompasses most traditional evolutionary algorithms and excludes more recent developments that still conform to the definition of a quasi-evolutionary algorithm as defined above. Once again, a crossover rule of order one may be used, so that every quasi-evolutionary algorithm with a vacuous crossover rule is also an evolutionary algorithm.

A mutation operator is even more dependent on the search space and can be almost any distribution. The most common mutators, however, are Bernoulli mutation in binary spaces and Gaussian mutation in Euclidean space, with Cauchy distributions also used for Euclidean space. In discrete or combinatorial spaces, mutation distributions typical involve random structural operators.

First, consider Gaussian mutation in $X = \mathbb{R}^d$. The mean of the Gaussian is simply the point being mutated (t^{-1}) and the covariance is a function of the prior points evaluated, often a constant. Then *Gaussian mutation* with a covariance-generating function Σ is given by

$$\mathcal{N}\left<\Sigma\right>[t, f] \quad = \quad \mathcal{N}\left(t^{-1}, \Sigma(H(t))\right), \tag{4.18}$$

where $\mathcal{N}(\mu, \Sigma)$ is the normal distribution and the symbol $\mathcal{N}$ is overloaded to represent Gaussian mutation as well.

When the search space is binary, $X = \{0,1\}^d$, *Bernoulli mutation* at rate p is given by

$$\mathcal{B}\left<p\right>[t \cup z, f](\{y\}) \quad = \quad \prod_j p^{|y_j - z_{i,j}|}(1-p)^{(1-|y_j - z_{i,j}|)}. \tag{4.19}$$

Bernoulli mutation is the standard mutation for genetic algorithms with binary encodings, whereas Gaussian mutation is the standard mutation for real vector encodings.

Putting all of these pieces together, a basic genetic algorithm with single-point crossover, proportional selection, and a binary encoding can be written as

$$\mathcal{SGA}\left<p\right> = (\mathcal{PS} \star ((\lhd \mathcal{PS}) \star \mathcal{SC})) \star \mathcal{B}\left<p\right>, \tag{4.20}$$

which is Goldberg's simple genetic algorithm with a mutation rate of p [77]. A common choice of genetic algorithm for searching in Euclidean space is to use ranking selection with uniform crossover and Gaussian mutation, namely,

$$\mathcal{RGA}\left<q, \sigma\right> = (\mathcal{RS}\left<q\right> \star ((\lhd \mathcal{RS}\left<q\right>) \star \mathcal{UC})) \star \mathcal{N}\left<\sigma I\right>, \tag{4.21}$$

where $q \in [1, 2]$ is the selection pressure and σ is a small constant rate of mutation. In both cases, the resulting algorithms are formally evolutionary algorithms, since they are composed of a population-Markov selection rule, a recombination operator with masked crossover, and a mutation operator.

Most standard genetic algorithms can be written down by mixing and matching the components described in the section along with domain-specific mutation operators, as was done in Equations 4.20 and 4.21. More esoteric genetic algorithms could also be represented in similar fashion with the definition of additional components.

4.2.4 Evolution Strategies

Evolution strategies differ from genetic algorithms primarily in the choice of selection and mutation operators, and in the fact that crossover is rarely used in evolution strategies, and is used with different crossover rules if so. Additionally, traditional evolution strategies also adapt their algorithm parameters dynamically. Standard evolution strategies are denoted as either (μ, λ)–ES or $(\mu + \lambda)$–ES. In this notation, μ is the number of parents, and λ is the number of children. The parents are always the μ best members of the last population, so if $K = 10$ and $\mu = 3$, then the parents are the top three members of the last population by fitness. A $(\mu + \lambda)$–ES has population size $K = \mu + \lambda$, and in each generation, the parents are retained unchanged from the prior generation, and λ new solutions are sampled from the parents. A (μ, λ)–ES has a population size $K = \lambda > \mu$; it discards the parents and replaces them with the children at each time step. The simplest evolution strategy is the $(1 + 1)$–ES, which is equivalent to simulated annealing at zero temperature. The most commonly used is probably the $(10, 100)$–ES, which tends to find solutions to basic benchmark problems with reasonable speed and accuracy.

Selection in evolution strategies first sorts the prior population by rank and then selects the next population. Ranking is performed by

$$\mathcal{R}\left[t, f\right]\left(\{y\}\right) = \delta_{k(t)}\left(R(y, f, H(t)^{-1}) + 1\right). \tag{4.22}$$

The λ children are then selected uniformly from among the top-ranked μ members of the last population in one of two ways.

$$\begin{aligned} \mathcal{U}\left\langle \mu\right\rangle[t,f](\{y\}) &= \frac{1}{\mu}\sum_{i=1}^{\mu}\delta_{H(t)^{-1,i}}(y) & (4.23)\\ \mathcal{U}_{+}\left\langle \mu\right\rangle[t,f](\{y\}) &= \begin{cases} H(t)^{-1,k(t)} & \text{if } k(t)\leq\mu \\ \mathcal{U}\left\langle \mu\right\rangle[t,f](\{y\}) & k(t)>\mu \end{cases} & (4.24)\end{aligned}$$

The selection rule $\mathcal{U}$ is used for so-called "comma" selection, where the parents are discarded and only the children remain in the population. The alternative version $\mathcal{U}_{+}$ is for "plus" selection, where both parents and children remain in the new population. Selection in evolution strategies is given by $\mathcal{ESS}\left\langle \mu\right\rangle \equiv \mathcal{R}\star\mathcal{U}\left\langle \mu\right\rangle$ for "comma" selection, and $\mathcal{ESS}_{+}\left\langle \mu\right\rangle \equiv \mathcal{R}\star\mathcal{U}_{+}\left\langle \mu\right\rangle$ for "plus" selection. This two-part decomposition is somewhat arbitrary. While $\mathcal{ESS}\left\langle \mu\right\rangle$ and $\mathcal{ESS}_{+}\left\langle \mu\right\rangle$ are well-defined, unique elements in $\mathcal{PBO}_K$ for a given λ, the decomposition $\mathcal{R}\star\mathcal{U}$ is just one way of expressing it, just as $1+3$ and $2+2$ are two different ways of expressing 4.

Evolution strategies often do not recombine selected points, but when they do, they often use higher-order crossover rules. The resulting algorithms are termed either as a $(\mu/\rho+\lambda)$–ES or a $(\mu/\rho,\lambda)$–ES, where ρ is the order of the crossover. Two crossover rules are commonly used: intermediate crossover and dominant crossover. Dominant crossover is a higher order generalization of uniform crossover to ρ parents. Intermediate crossover averages the parent components. Like the crossover methods used for genetic algorithms, these two methods assume that the search space has a product space structure. Additionally, intermediate crossover requires that the search space be a vector space with addition and scalar multiplication. Intermediate crossover is easy to express as a point distribution on the average of the parents. If

$$\text{average}(t) = \frac{1}{\rho}\sum_{i=1}^{\rho} H(t)^{-i,k(t)}, \tag{4.25}$$

then intermediate crossover is given by

$$\mathcal{IC}\left\langle \rho\right\rangle[t,f]\left(\{y\}\right) = \delta_{\text{average}(t)}\left(y\right). \tag{4.26}$$

Dominant crossover can be expressed by generalizing the idea of crossover masks so that the mask ranges from 1 to ρ, i.e. $M \in \{1,\ldots,\rho\}^{d}$. Further, let $p(i,t)$ be the i^{th} parent on the trajectory t with order ρ, $p(i,t) =$

parents$(t, \rho, k(t), K)^{-i}$. Also, let $M \otimes_i p(i,t)$ denote the point that is zero for each component where $M \neq i$ and equal to $p(i,t)$ when $M = i$. Then the crossover random variable can be redefined to

$$C_t = \sum_{i=1}^{\rho} M \otimes_i p(i,t). \tag{4.27}$$

And then if $\mathbb{P}_M = \text{Uniform}\left(\{1, \dots, \rho\}^d\right)$, dominant crossover is given by

$$\mathcal{DC}\left<\rho\right>[t, f](A) = \sum_{z \in \{1,\dots,\rho\}^d} \mathbb{P}\left(C_t \in A \mid M = z\right) \, \mathbb{P}_M(z) \tag{4.28}$$

The final element is mutation. As with genetic algorithms, the type of mutation is customized to the search space, and sometimes to the objective. Most often, evolution strategies are employed in $\mathbb{R}^d$, and Gaussian mutation is used. One of the main differences between genetic algorithms and evolution strategies, however, is the level of effort expended to adapt the mutation parameters over the course of optimization. Sometimes, this adaptation occurs at a global level, so that all elements of the next population are generated from the same mutation distribution. In this case, let $\Sigma(t)$ be a matrix-valued function that takes a trajectory and performs trajectory-specific computations to produce a covariance matrix for mutation (see e.g [27, 85]). Then the $(\mu/\rho, \lambda)$–ES with dominant crossover and Gaussian mutation is an element of $\mathcal{PBO}_\lambda\left[\mathbb{R}^d\right]$ and can be written as

$$\mathcal{ES}\left<\mu,\rho\right> = \mathcal{ESS}\left<\mu\right> \star \left(\triangleleft\mathcal{ESS}\left<\mu\right> \star \left(\cdots \star \left(\triangleleft_{\rho-1}\mathcal{ESS}\left<\mu\right> \star \mathcal{DC}\left<\rho\right>\right)\right)\right) \star \mathcal{N}\left<\Sigma(t)\right> \tag{4.29}$$

Notice that this equation identifies evolution strategies as a formal evolutionary algorithm, since it is the convolution of a selection rule, a recombination operator with masked crossover, and a mutation operator. Notably, this method fails to meet the formal definition of a genetic algorithm. The covariance function includes some information about previous populations, so this evolution strategies method is not population-Markov. Also, if intermediate crossover had been used, then the algorithm would only be a quasi-evolutionary algorithm by the definitions above. As mentioned, crossover is a relatively recent innovation in evolution strategies, and so this violation may be regarded as marking a early trend towards the quasi-evolutionary algorithms analyzed below.

The characterization in this subsection applies to general evolution strategies, but does not apply to CMA-ES. Apart from their reliance on a trajectory-specific covariance matrix $\Sigma(t)$, modern versions of CMA-ES have more in common with EDAs than with traditional evolution strategies, and these methods are handled together in Section 4.3.3.

Rather than having a global adaptation procedure, evolution strategies often adapt mutation parameters with each point. In this case, the mutation parameters are carried along with the selected point. The point itself is mutated using the current mutation parameters, and then the mutation parameters are themselves mutated using a global adaptation scheme. Adaptive mutation of this form cannot be represented in $\mathcal{PBO}_K\left[X\right]$. However, if the mutation parameters range over a space Θ, then adaptive mutation can be described on an extended state space as an element of $\mathcal{PBO}_K\left[X \times \Theta\right]$, where $X \times \Theta$ is the Cartesian product. The objective function can be extended to this product space by defining $\tilde{f}(x, \theta) = f(x)$. Finally, since X can be embedded in $X \times \Theta$, elements of $\mathcal{PBO}_K\left[X\right]$ can be projected trivially into $\mathcal{PBO}_K\left[X \times \Theta\right]$ using this embedding, and so adaptive evolutionary strategies can be compared directly with non-adaptive ones for theoretical and practical purposes. The issue of space extension will be explored further in Section 4.3.1.

4.2.5 The $(1+1)$–ES as the Norm Limit of Simulated Annealing

The standard theoretical example of an evolution strategy is the (1+1)–ES in $\mathbb{R}^d$. In terms of this dissertation, this optimizer is represented by

$$\mathcal{ES}_+\left\langle 1, 1\right\rangle\left[t, f\right] = \left\{ \begin{array}{cl} \mathcal{ESS}_+\left\langle 1\right\rangle & \text{if } k(t) = 1 \\ \mathcal{ESS}_+\left\langle 1\right\rangle \star \mathcal{N}\left\langle \sigma(t)^2 I\right\rangle & \text{if } k(t) = 2 \end{array}\right. \tag{4.30}$$

It operates on a population of size two. The first member of the population is always the current best solution, and the second member is a proposed replacement mutated from the current best. The function $\sigma(t)$ is a globally adaptive parameter that controls the standard deviation of mutation. The standard deviation is controlled so that it improves approximately 23% of the proposed solutions.

The description of the $(1+1)$–ES is reminiscent of simulated annealing. There is an accepted solution and a proposed solution. The proposed solution is generated from the acceptance probability using Gaussian variation. The standard deviation of the Gaussian distribution is controlled so that the running best has a 0.23 probability of being replaced. The only notable difference is that the $(1+1)$–ES lacks an explicit acceptance probability. In fact, by gradually reducing the cooling schedule, the $(1+1)$–ES can be shown to be the norm-limit of $\mathcal{SA}$. While this fact has always been obvious to the intuition, the formalisms proposed in this dissertation allow it to be proven as mathematical fact; without the formalism, the result could not be clearly achieved.

To obtain this result, redefine simulated annealing using the terminology of the last several sections, especially that of Section 4.1.5:

$$\mathcal{SA}\left\langle T\right\rangle [t,f] = \begin{cases} \mathcal{B}\langle A(t^{-1},t^{-2},f,T(|t|/2)),t^{-1},t^{-2}\rangle & \text{if } k(t)=1 \\ \mathcal{N}\left\langle \sigma(t)I\right\rangle & \text{if } k(t)=2, \end{cases} \tag{4.31}$$

where $T:\mathbb{N}\to\mathbb{R}$ is a cooling schedule for simulated annealing.

Theorem 4.2.3. *Suppose T_n is a cooling schedule such that $T_n \to 0$ as $n\to\infty$. Then $||\mathcal{SA}\left\langle T_n\right\rangle - \mathcal{ES}_+\left\langle 1,1\right\rangle||_{\mathcal{MF}} \to 0$. That is, the $(1+1)$–ES is the limit of simulated annealing using the norm of $\mathcal{PF}$, assuming both optimizers use the same initial distribution.*

Proof. Assume that the initial population is generated from the same initial distribution. Fix the objective function f and the trajectory t with $|t|>2$ to represent any non-initial state. First, note that if $k(t)=2$, then $\mathcal{SA}\left\langle T_n\right\rangle = \mathcal{ES}_+\left\langle 1,1\right\rangle$ for all n. Thus the interesting case occurs when $k(t)=1$. Consider the sequence of acceptance probabilities $A_n(t) = A\left(t^{-1},t^{-2},T_n(|t|/2)\right)$. As $T_n\to 0$, either $A_n(t)\to 0$ if $f(t^{-2})>f(t^{-1})$ or $A_n(t)\to 1$ otherwise. Let

$$\tilde{\mathcal{B}}_n[t,f] = \mathcal{B}\left\langle A_n(t),t^{-1},t^{-2}\right\rangle [t,f],$$

When $k(t)=1$, then $\mathcal{SA}\left\langle T_n\right\rangle = \tilde{\mathcal{B}}_n$, and for $B\in\mathcal{B}_\tau$,

$$\mathcal{ES}_+\left\langle 1\right\rangle [t,f](B) = \mathcal{ESS}_+\left\langle 1\right\rangle [t,f] = \begin{cases} 1_B(t^{-1}) & \text{if } f(t^{-1})\geq f(t^{-2}) \\ 1_B(t^{-2}) & \text{otherwise,} \end{cases}$$

where 1_B is the indicator function for the set B.

If neither t^{-1} nor t^{-2} are in B, then

$$\mathcal{SA}\left\langle T_n\right\rangle[t,f](B) = \mathcal{ES}_+\left\langle 1\right\rangle[t,f](B) = 0.$$

If both are in B, then $\mathcal{SA}\left\langle T_n\right\rangle[t,f](B) = \mathcal{ES}_+\left\langle 1\right\rangle[t,f](B) = 1$. Thus the only possible differences are realized when B contains only one of the points. It suffices to let B contain only t^{-1} and not t^{-2}, since convergence on B implies convergence on $X \setminus B$ as a consequence. There are now two cases.

Case 1: $f(t^{-1}) \geq f(t^{-2})$. In this case, $A_n(t) \to 1$, so $\tilde{B}_n \to 1$, since B contains t^{-1}. So

$$\left|\mathcal{SA}\left\langle T_n\right\rangle[t,f](B) - \mathcal{ES}_+\left\langle 1\right\rangle[t,f](B)\right| = \left|\tilde{B}_n[t,f](B) - 1\right| \to 0. \tag{4.32}$$

Case 2: $f(t^{-1}) < f(t^{-2})$. In this case, $A_n(t) \to 0$, so $\tilde{B}_n \to 0$, since B does not contain t^{-2}. So

$$\left|\mathcal{SA}\left\langle T_n\right\rangle[t,f](B) - \mathcal{ES}_+\left\langle 1\right\rangle[t,f](B)\right| = \tilde{B}_n[t,f](B) \to 0. \tag{4.33}$$

Thus in either case, since t, f and B were arbitrary, then for $\epsilon > 0$ there exist t^* and B^* such that for n large,

$$\begin{aligned} \left|\left|\mathcal{SA}\left\langle T_n\right\rangle - \mathcal{ES}_+\left\langle 1\right\rangle\right|\right| &< \left|\mathcal{SA}\left\langle T_n\right\rangle[t^*,f](B^*) - \mathcal{ES}_+\left\langle 1\right\rangle[t^*,f](B^*)\right| + \epsilon \\ &< 2\epsilon, \end{aligned} \tag{4.34}$$

and so the proof holds. $\square$

This simple theorem justifies the approach of unifying optimization algorithms through the formalization of $\mathcal{PF}$ because it allows a rigorous comparison of two typologically distinct optimization methods, one a Monte Carlo method and the other an evolutionary method. It also helps to identify opportunities for new algorithms, as is done with evolutionary annealing in Chapter 11.

4.3 Quasi-Evolutionary Algorithms

The previous section explored the relationship of evolutionary algorithms to the proposed formalism. This section examines how some of the natural computation and quasi-evolutionary methods from Chapter 2 can be formalized.

4.3.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is used to search rectangular regions within $\mathbb{R}^d$ [62, 107]. To generalize the algorithm somewhat, the formalization will assume that the search space X is a closed rectangular subset of a vector space. Arbitrary bounded subsets of a vector space (say, $\Omega \subseteq X$) can be searched in this way through the use of a feasibility region by setting $f(x) = \infty$ for $x \notin \Omega$ [111].

PSO maintains a population of particles metaphorically representing a flock of birds. Each particle has a position and a velocity in the search space, and the particles are ranked by fitness. As with differential evolution, the population may be viewed as a sequence of slots. PSO stores the best overall solution (called the *global best*) as well as the best solution that has occurred at each slot over all prior populations (the *local best*). The position and velocity are often initialized uniformly at random over the search space. At each generation, the velocity of each particle is shifted towards the position of both the global best and the local best. The position of the particle is then updated according to the velocity. For the i^{th} slot of the population, the update rules for the velocity v^{n+1} and the position x^{n+1} after the n^{th} generation are

$$v_i^{n+1} = \omega v_i^n + \phi_p U_{p,i}^n x_{p,i}^n + \phi_g U_{g,i}^n x_g^n \tag{4.35}$$

$$x_i^{n+1} = x_i^n + v_i^{n+1}, \tag{4.36}$$

where ω decays the current velocity; $x_{p,i}^n$ and x_g^n are the local and global best positions at the n^{th} generation, respectively; ϕ_p and ϕ_g control the sensitivity to the local best and global best solutions; and $U_{p,i}^n, U_{g,i}^n$ are uniform random variables on $[0, 1]$. If the velocity v_i^{n+1} will cause the particle to exit the search space, then the velocity can be clipped so that the position x_i^{n+1} is on the boundary of X. Another approach is to allow particles to exit the rectangular boundary, but to draw them back in by using the feasibility modification to the objective function. Feasibility guarantees that x_g^n and $x_{p,i}^n$ are inside of X, and as long as $|\omega| < 1$, the particle will eventually be pulled back into the search space.

The three parameters ω, ϕ_p and ϕ_g control the algorithm and are arbitrary. Studies have shown that if $\phi_p = 0$, then the optimal settings for ω and ϕ_g over an average of benchmarks occur when $\omega \in [-0.6, 0]$ and $\phi_g \in [2, 4]$ [153]. The use of negative values for the velocity decay ω wreaks havoc with the flock analogy but is nonetheless effective.

To formalize PSO, the search space must be extended to include the velocities. Otherwise, there is no way to infer the velocity from the particle positions, since the velocity is initialized randomly. To this end, let the extended search space by $Y = X \times X$ with an altered objective $\tilde{f}(x, v) = f(x)$. If the initial velocity is set to zero, as is sometimes done, then the velocities may be inferred from the particle trajectories and this extension is not necessary.

Let $\mathcal{PSO}\left<\omega, \phi_p, \phi_g, K\right>$ be an instantiation of PSO with the specified parameters and population size K. The optimizer $\mathcal{PSO}$ will be defined by constructing a process Z on Y and then setting $\mathcal{PSO}$ to be the distribution of this process so that $Z \sim \mathcal{PSO}\left<\omega, \phi_p, \phi_g, K\right>$. Let the vectors u and ℓ represent the upper and lower bounds for X. Use the superscript n, i to represent the state of the i^{th} slot in the n^{th} population, i.e. $Z^{n,i} = Z_{(n-1)K+i+1}$. Initialize the process so that $Z^{0,i} \sim \text{Uniform}\left([\ell, u]^2\right)$. At every step, the process consists of a position and a velocity component, $Z^{n,i} = (X^{n,i}, V^{n,i})$. Let the update rules in Equation 4.35 govern the position variable $X^{n,i}$ and the velocity $V^{n,i}$, with variables $X_{p,i}^n$ and X_g^n to represent the local and global best. Then $\mathcal{PSO}$ is given by

$$\mathcal{PSO}\left<\omega, \phi_p, \phi_g, K\right>[t, f](A) = \mathbb{P}\left(Z^{|H(t)|,(|t| \operatorname{mod} K)} \in A \mid Z_n = t^n \, \forall n < |t|\right) \tag{4.37}$$

$\mathcal{PSO}$ can be reduced to the search space X by marginalizing out the velocities when required in order to compare PSO with other methods.

Formally, PSO is a quasi-evolutionary algorithm by the definition above with a vacuous mutation operator. PSO can be characterized as using three selection rules, one that selects the previous position and velocity, one that selects the global best, and one that selects the local best. The crossover rule then applies Equation 4.35 to these three items to generate the recombined point. Since the recombined point is also the output, the mutation operator must be vacuous. Thus PSO introduces a unique and complex crossover rule, but is formally a quasi-evolutionary algorithm. It is not an evolutionary algorithm, since there is no masked crossover rule that implements PSO. It is also important to note that PSO is not population-Markov, since it depends on the global and local best solutions at each time step.

4.3.2 Differential Evolution

Like PSO, differential evolution is designed to search a rectangular region in $\mathbb{R}^d$. Once again, this method can be easily generalized to any search domain that is a closed rectangular subset of a vector space. Suppose that the search space X is a subset of a finite-dimensional vector space defined by a closed rectangular region with upper and lower boundaries u and ℓ.

As mentioned in Chapter 2, Ghosh et al. [74] recently proved that differential evolution converges to the true global optimum on functions with two continuous derivatives. The techniques employed by Ghosh et al. are similar to the techniques employed in this dissertation, and thus their result provides further evidence that theorems about stochastic optimizers can be stated and proven in a suitable formalization.

Differential evolution passes through three phases: selection, mutation, and crossover. Notably, crossover in differential evolution crosses a mutated vector with the selected vector, and so differential evolution does not meet the formal definition of an evolutionary algorithm.

Differential evolution uses a form of selection that will be termed *local best selection*. This method selects the best member of the trajectory so far along each component. Let $b_i(t) \in X$ be the best solution found so far at the i^{th} slot along the trajectory t, $b_i(t) = \text{argmin}_{t^{n,i}} f\left(t^{n,i}\right)$, where the superscripts indicate the i^{th} member of the n^{th} population, as in the description of PSO above. Local best selection always selects $b_i(t)$ for the i^{th} member of the population, i.e. with population size K it is the point distribution given by

$$\mathcal{LBS} < K > [t, f]\,(A) = \mathbb{P}\left(b_{|t| \bmod K}\,(t) \in A\right). \tag{4.38}$$

Mutation in differential evolution adds the difference between two members of a population to a third member of the population. The vector to which the difference is added is termed the *donor vector*, and K donor vectors are chosen from among the local best vectors in one of three ways. In random mutation, each donor vector may be a chosen uniformly at random from among the local best vectors. In target-to-best mutation, every donor vector may be fixed as the global best vector (denoted as t^* for a trajectory t). In best mutation, the i^{th} donor vector may be chosen as a particular point along the line from the i^{th} local best to the the global best vector. Once the donor vector is selected, then two other distinct vectors are chosen randomly from among the local best vectors, and the donor vector is moved in direction of their difference, multiplied by a weighting factor denoted F. These possibilities are expressed as

$$\begin{aligned} Y_{\text{rand},i}(t) &= b_{R_1}(t) + F\left(b_{R_2}(t) - b_{R_3}(t)\right) && (4.39)\\ Y_{\text{target},i}(t) &= b_i(t) + F\left(t^* - b_i(t)\right) + F\left(b_{R_1}(t) - b_{R_2}(t)\right) && (4.40)\\ Y_{\text{best},i}(t) &= t^* + F\left(b_{R_1}(t) - b_{R_2}(t)\right), && (4.41) \end{aligned}$$

where R_1, R_2, and R_3 are distinct uniformly random indices between 1 and d, inclusive, and d is the dimension of the space. Some versions also add a second difference chosen randomly from among the remaining local best vectors. Let Y_* stand for any one of Y_{rand}, Y_{target}, or Y_{best}. Then mutation in differential evolution can be represented by

$$\mathcal{DM}_* < F, K > [t, f](A) = \mathbb{P}\left(Y_{*,|t| \bmod K}(t) \in A\right). \tag{4.42}$$

Differential evolution recombines the local best vectors with the mutated vectors to create the next population. The two crossover strategies are termed *binomial* and *exponential*. Both schemes can be described using crossover masks, and each is parameterized by a *crossover rate* denoted by CR. Binomial crossover is so named because each mutated component is selected as a Bernoulli trial with probability CR, i.e. $\mathbb{P}\left(M_i = 0\right) = CR$, recalling that

$M_i = 0$ implies that the "mother" (the mutated vector) is chosen. However, if $M_i = 1$ for all i, the sample is rejected, so that at least one mutated component is included. Exponential crossover copies a subsequence of adjacent components from the mutated vector onto the local best vector. A random index I in $1, \ldots, d$ is chosen along with a random length L also in $1, \ldots, d$. Then $M_i = 0$ if $i \geq I$ and $i < I + L$, applying modular arithmetic as necessary.

Recalling the crossover mask rule $\mathcal{C} < \mathbb{P}_M >$ and letting $M[CR]$ be the selected crossover mask, differential evolution is thus given by

$$\mathcal{DE}_* < F, CR, K > = \mathcal{LBS} < K > \star \left(\mathcal{DM}_* < F, K > \star \mathcal{C} \left\langle \mathbb{P}_{M[CR]} \right\rangle \right). \quad (4.43)$$

As with PSO, DE is also a quasi-evolutionary algorithm according to the formal definition in this chapter. The convolution $\mathcal{DM}_* \star \mathcal{C}$ fits the definition of a fourth-order crossover rule, provided that the vectors t^*, b_i, b_{R_1}, b_{R_2}, and b_{R_3} are selected by a selection rule. The mutation operator for DE is vacuous, as it is for PSO. DE is not an evolutionary algorithm, since it does not use a masked crossover rule. Also, DE is not population-Markov due to its use of the local best solutions. Indeed, PSO and DE share quite a few structural similarities, and it is interesting that the proposed formalism draws them out.

4.3.3 Parameterized Quasi-Evolutionary Methods

Both Estimation of Distribution Algorithms and Natural Evolution Strategies have an explicit representation as a parameterized probabilistic model. These methods thus fit into the formalism naturally. Each of them is described by a distribution $\pi\left(dx \mid \theta\right)$ where θ is drawn from a parameter space. The parameters θ are reestimated once per generation based on the population history, so that $\theta = \hat{\theta}(H(t), f)$ to reflect the dependence of the parameters on the prior populations and their evaluations. Then all EDA and NES instances can be represented by

$$\mathcal{G}\left\langle \pi, \theta \right\rangle [t, f](A) = \int_A \pi\left(dx \mid \hat{\theta}(H(t), f) \right) \quad (4.44)$$

for an appropriate model π and a parameter estimation function $\hat{\theta}$.

Also, both EDAs and NES are quasi-evolutionary algorithms according to the formalism above. EDAs use truncation selection to choose the best M members of the prior population. These points are then used to build a probabilistic model. This model depends only on the selected points and not on their objective value, so the construction and sampling of the model are objective agnostic. Therefore, the model-sampling process can also be described formally as a crossover rule of order M. Thus the EDA consists of truncation selection, recombination through model-sampling, and vacuous mutation.

NES and CMA-ES use truncation selection in the same way as traditional evolution strategies. Just like EDAs, they then build a probabilistic model from the selected points. However, these methods differ from EDAs in that they maintain a set of global parameters that are adjusted to follow the gradient of certain metaparameters. Thus NES implementations are evolutionary algorithms for the same reason that EDAs are, but EDAs are population-Markov, unlike NES. Neither NES nor EDAs are strict evolutionary algorithms, since the model-building process cannot be implemented as a masked crossover rule.

4.3.4 Non-Quasi-Evolutionary Algorithms

Under analysis, each of the quasi-evolutionary algorithms studied in this section has formally satisfied the definition of a quasi-evolutionary algorithm proposed in this chapter. This result is not surprising, since all of these methods were ultimately inspired by previous work on evolutionary algorithms. However, it does beg the question of whether a practical trajectory-restricted optimizer exists that is not formally a quasi-evolutionary algorithm. Under further consideration, simulated annealing, generating set search, and even Nelder-Mead[4] are also formally quasi-evolutionary algorithms by the criterion above.

---

[4] It is difficult to cast Nelder-Mead into the format of a selection rule, recombination and mutation. To see how it might be done in $d+1$ dimensions, $2d+2$ selection rules can be used, with $d+1$ rules selecting the simplex endpoints in some fixed order and an additional $d+1$ points selecting the same endpoints in rank order. The if-then rules of Nelder-Mead can then be implemented within an objective-agnostic crossover rule.

At least one trajectory-restricted algorithm from Chapter 2 is not a quasi-evolutionary algorithm even by this definition. Gradient-based methods with estimated gradients are trajectory-restricted, but depend on the raw objective values of more than one point. Since a selection rule can only choose one previously observed point, it cannot encode the estimated gradient, which depends on at least two points, and since a crossover rule and mutation operator must be objective agnostic, they cannot compute the gradient either. Thus optimizers that are not quasi-evolutionary algorithms do exist under this definition.

It is important to consider how to distinguish between algorithms that fall traditionally within the ambit of evolutionary computation from those that do not. If the population size K were required to be greater than one, some versions of generating set search would still be included, since they can generate a fixed number of points that can be evaluated in parallel. Further, the $(1+1)$–ES, a traditional evolutionary algorithm, would be excluded under a certain interpretation and included under another. A quasi-evolutionary algorithm could be defined as factorial or homogeneous, but then one or more evolutionary algorithms would be excluded.

When defining evolutionary algorithms, only masked crossover rules were allowed, which reflect traditional intuitions about evolutionary computation. As a positive effect, PSO, DE, EDAs, and NES fail to meet the formal definition of an evolutionary algorithm under this assumption. But intermediate crossover is excluded by this definition, and along with it some evolution strategies and even certain forms of neuroevolution, such as NEAT. And yet even this definition is more inclusive than some might prefer, since other optimizers not traditionally included within evolutionary computation could be expressed as evolutionary algorithms without crossover, such as simulated annealing.

In the final analysis, it is not important to draw a strong distinction between which algorithms are and are not evolutionary or quasi-evolutionary algorithms. A formal analysis such as the one undertaken in this dissertation actually serves to undermine such categorical schemes, as the reasoning in the prior paragraph shows. In fact, it is a major benefit that the formal setting removes the ability to distinguish certain methods categorically, because it reveals important similarities among these methods along several dimensions that would not be evident otherwise.

4.4 Conclusion

Population-based optimizers were reviewed in this chapter to show how these optimizers fit into the formalism adopted in this dissertation. This exercise demonstrated that methods as diverse as genetic algorithms, evolution strategies, particle swarm optimization, differential evolution, and estimation of distributions algorithms all fit within the proposed framework for formal analysis. In addition, some of the advantages of this approach were demonstrated by proving that the $(1 + 1)$–ES method is the limit of simulated annealing with respect to the optimizer norm.

This discussion has also proposed a definition for evolutionary and quasi-evolutionary algorithms that requires selection, recombination, and mutation, all explicitly defined. These definitions effectively distinguished traditional evolutionary algorithms from their more recent quasi-evolutionary variants, but it was not possible to draw a categorical distinction between quasi-evolutionary algorithms and non-evolutionary methods such as Nelder-Mead and generating set search. Indeed, it is not clear that such a distinction is inherently useful, since this formalism aims to provide a single setting within which all of these algorithms can be compared. It is a confirmation of this approach that several algorithms proposed with distinct motives and inspirations bear structural similarities to each other that can be made clear using the tools provided by this analysis.

The following chapters will not treat population-based optimizers separately from other optimizers, but the equations and formulae in this chapter demonstrate effectively that the subsequent results apply equally to evolutionary and quasi-evolutionary optimization methods. The next chapter discusses the continuity of various optimizers, and the chapter after next addresses the integrability of the optimization trajectory. These results begin an analysis that will culminate in extended No Free Lunch theorems for optimization and an explicit definition of the duality between optimizers and random test procedures in Chapters 9 and 10.

Chapter 5

Continuity of Optimizers

The two previous chapters presented a formalization of stochastic optimizers. Subsequent chapters will leverage this formal setting to show theoretically and experimentally how different optimizers perform and how they use information. In order to obtain these results, this chapter and the next one will develop the necessary analytic tools. This chapter focuses on continuity, one of the primary tools of analysis in general. Continuity considers the question of whether it is reasonable to assume that the output of an optimizer will be similar when the inputs are similar. In the following chapters, it will be important to know when optimizers are continuous, and the theorems in the chapter provide the tools for answering this question. In particular, it will be shown that most non-deterministic optimizers are continuous on a broad range of trajectories and objectives.

5.1 Background and Motivation

This chapter studies the continuity of the one-step optimizers from Chapter 3. These optimizers were defined as functions from a trajectory and an objective function to a signed measure over the search space. In this context, there are two aspects of continuity that must be considered:

- Given similar evaluation trajectories, will an optimizer $\mathcal{G}$ choose similar evaluation points?
- Given similar objective functions, will $\mathcal{G}$ make similar decisions?

The first question pertains to continuity in the trajectory, and the second question to continuity in the objective function. If both questions can be answered affirmatively, then $\mathcal{G}$ is jointly continuous, or simply continuous.

Continuity is a central topological concept. In topological terms, a function is continuous if it maps open sets into open sets. An open set is nothing more than a set that is declared to be open by the topology. A topology is in fact defined by the sets that it declares to be open, and the open sets are arbitrary within certain consistency constraints. Thus continuity of a function is always continuity with respect to a particular topology on each of the input and output spaces.

The most familiar type of topology is the metric topology, which induces the epsilon-delta definition of continuity. Under a metric topology, a set A inside of a metric space X with metric d is open if for every point $x \in A$ there is some $\epsilon > 0$ such that $d(x, y) < \epsilon$ implies $y \in A$ for all $y \in X$. A function f that maps one metric space (X, d_X) to another metric space (Y, d_Y) is continuous if for every $\epsilon > 0$ and every point x there exists a $\delta = \delta(x) > 0$ such that for all y with $d_X(x, y) < \delta$, it holds that $d_Y(f(x), f(y)) < \epsilon$.

In Chapter 3, $\mathcal{MF}$ was introduced as a space of functions from trajectories and objectives to finite signed measures. In order to address continuity of optimizers, a topology needs to be specified for the input and output spaces. The output space is the space of finite signed measures, $\mathcal{M}[X]$. The input space is the Cartesian product of two spaces: the space of trajectories, $\mathcal{T}[X]$ and the space of objectives, $\mathbb{R}^X$. On a fixed objective, an optimizer can be viewed as a function from $\mathcal{T}[X]$ to $\mathcal{M}[X]$. On a fixed trajectory, an optimizer is a function from $\mathbb{R}^X$ to $\mathcal{M}[X]$. If neither parameter is fixed, then the optimizer is a function from $\mathcal{T}[X] \times \mathbb{R}^X$ to $\mathcal{M}[X]$. Whether or not an optimizer is continuous depends on the topology assigned to each one of $\mathcal{M}[X]$, $\mathcal{T}[X]$, $\mathbb{R}^X$, and $\mathcal{T}[X] \times \mathbb{R}^X$.

The search space X was assumed to be a Hausdorff topological space with topology τ. The topology τ can be extended to create a topology on $\mathcal{T}[X]$. A suitable topology for $\mathcal{T}[X]$ can be generated from a base of arbitrary Cartesian products over open sets in τ, i.e.

$$\mathcal{O} = \left\{ O \subseteq \mathcal{T}[X] \mid \exists n < \infty \; s.t. \; O = O_1 \times \cdots \times O_n \text{ with } O_i \in \tau \;\; \forall 1 \leq i \leq n \right\}. \tag{5.1}$$

The standard topology on $\mathcal{T}[X]$ is assumed to be the smallest topology on $\mathcal{T}[X]$ in which every set in $\mathcal{O}$ is open. If τ is a metric topology with a metric

ρ on X, a metric on $\mathcal{T}[X]$ is given by

$$d_\rho(t_1, t_2) = \left| |t_1| - |t_2| \right| + \sum_{i=1}^{|t_1| \wedge |t_2|} \rho\left(t_1^i, t_2^i\right), \tag{5.2}$$

where t_1^i is the i^{th} element of t_1, and $|t_1|$ is the length of the trajectory t_1. In this chapter, X is generally assumed to be metric; the metric topology on $\mathcal{T}[X]$ generated by d_ρ is therefore treated as the standard topology on $\mathcal{T}[X]$.

The space $\mathbb{R}^X$ is a topological space under the *topology of pointwise convergence*, for which a sequence of functions $\{f_n\}_{n\in\mathbb{N}}$ converges to a function f if $\lim f_n(x) = f(x)$ for all $x \in X$. The topology of pointwise convergence admits a metric in only limited circumstances.[1] The basic open sets of this topology are intervals bounded on either side by functions. That is, for $f, g \in \mathbb{R}^X$, define $f < g$ to mean that $f(x) \leq g(x)$ for all $x \in X$ and there exists at least one $x_0 \in X$ such that $f(x) < g(x)$. The interval $I[f, g]$ is defined by $I[f, g] = \{h \in \mathbb{R}^X \mid f < h < g\}$, and the topology of pointwise convergence is the smallest topology containing every interval on $\mathbb{R}^X$. A function G on $\mathbb{R}^X$ is continuous if $G(f_n) \to G(f)$ whenever $f_n \to f$ pointwise. Other topologies on $\mathbb{R}^X$ are possible, including a metric topology based on the extended metric $d(f, g) = \sup_{x\in X} |f(x) - g(x)|$. Such topologies are not explored here, since pointwise convergence is sufficient for the purposes of this chapter.

The space $\mathcal{M}[X]$ is a normed vector space as discussed in Chapter 3. Every norm induces a metric given by $d(x, y) = ||x - y||$. The metric topology produced by the norm-induced metric is referred to as the *norm topology*. The norm topology on $\mathcal{M}[X]$ will be utilized here as a default.

The definition of continuity adopted here is the topological one. For a trajectory t and an objective f, an optimizer $\mathcal{G}$ is continuous in objectives

---

[1] By Urysohn's metrization theorem, the product space $\mathbb{R}^X$ will be metrizable if it is Hausdorff, regular, and second countable [8]. In this text, $\mathbb{R}^X$ with the product topology is Hausdorff. In many cases, it will also be regular. In order for $\mathbb{R}^X$ to be second countable (i.e., the topology is generated from a countable family of sets), X would have to countable, since $\mathbb{R}$ is second countable. If X is only second countable and not countable, then the product topology has no countable base. Thus, for instance, if $X = \mathbb{R}^d$, then $\mathbb{R}^X$ is not metrizable. For an illustrative counterexample in $C[X]$, see [68].

at t, f if it maps open neighborhoods of f to open neighborhoods of $\mathcal{G}[t, f]$. Similarly, $\mathcal{G}$ is continuous in trajectories at t, f if it maps open neighborhoods of t to open neighborhoods of $\mathcal{G}[t, f]$. If $\mathcal{G}$ is continuous in objectives and continuous in trajectories at t, f, then it is *jointly continuous* at t, f. When $\mathcal{G}$ is described as simply *continuous in objectives* (or *continuous in trajectories*) without qualification, then it is intended to mean that $\mathcal{G}$ is continuous in objectives (or trajectories) everywhere in the space. The following two propositions translate the requirements for continuity into more familiar language, assuming X is a metric space.

Proposition 5.1.1. *An optimizer $\mathcal{G} \in \mathcal{MF}[X]$ is continuous in objectives at f if for any sequence of objectives $\{f_n\}$, $f_n \to f$ implies $||\mathcal{G}[t, f] - \mathcal{G}[t, f_n]||_{\mathcal{M}} \to 0$.*

Proposition 5.1.2. *An optimizer $\mathcal{G} \in \mathcal{MF}$ is continuous in trajectories at t if for every $\epsilon > 0$ there exists a $\delta > 0$ such that whenever $d_\rho(t, u) < \delta$ then $||\mathcal{G}[t, f] - \mathcal{G}[u, f]||_{\mathcal{M}} < \epsilon$.*

If the space X is not metric, then continuity in trajectories remains well-defined in the topological sense. For now, X will be assumed to be a metric space to simplify the arguments in this section, although most of the results do apply more generally.

The norm topology on $\mathcal{M}[X]$ is the most obvious choice for a topology on $\mathcal{M}[X]$. Other topologies are possible as well, but are not studied in this dissertation. Occasionally, in order to make the distinction between other forms of continuity and continuity derived from the norm topology of $\mathcal{M}[X]$, optimizers that are continuous under the norm topology of $\mathcal{M}[X]$ may also be referred to as *norm-continuous*.

If an optimizer is continuous in objectives, then it can be expected to perform similarly on similar problems. If an optimizer is continuous in trajectories, then it can be expected to make similar decisions on similar trajectories. The continuity of an optimizer is not important in itself. However, certain theorems can be formulated that apply only to continuous optimizers, and so it is important to know which optimizers are continuous. For example, in Chapter 7, the continuity of performance criteria will be studied, and it will be seen that performance criteria are continuous on continuous optimizers, meaning that continuous optimizers can be expected to have similar performance on similar problems. The remainder of this chapter develops definitions and theorems that establish the conditions under which common optimizers such as evolutionary algorithms, differential evolution, and stochastic gradient descent are continuous.

5.2 Deterministic Optimizers

Most deterministic optimizers are discontinuous in the norm topology. Given a deterministic optimizer $\mathcal{D}$ and an objective f, the asymptotic trajectory proposed by $\mathcal{D}$ on f is a unique sequence. Given a particular trajectory t, denote the unique next point by $p(t, f)$. Then $\mathcal{D}[t, f](dx) = \delta_{p(t,f)}(x)$. Choose an objective $g \neq f$, and then

$$\begin{aligned} \left|\mathcal{D}[t,f] - \mathcal{D}[t,g]\right|(X) &= \int \left|\mathcal{D}[t,f](dx) - \mathcal{D}[t,g](dx)\right| \\ &= \begin{cases} 0 & p(t,f) = p(t,g) \\ 2 & p(t,f) \neq p(t,g) \end{cases} . \end{aligned} \tag{5.3}$$

So if $p(t, f) \neq p(t, g)$, it follows that

$$||\mathcal{D}[t,f] - \mathcal{D}[t,g]||_{\mathcal{M}} \ \geq \ \left|\mathcal{D}[t,f] - \mathcal{D}[t,g]\right|(X) \ = \ 2. \tag{5.4}$$

Since g was arbitrary, it is clear that $\mathcal{D}$ cannot be continuous in objectives unless $p(t, f) = p(t, g)$ for all g sufficiently close to f. That is, if $f_n \to f$, then $||\mathcal{D}[t, f] - \mathcal{D}[t, f_n]||_{\mathcal{M}} \geq 2$ regardless of n whenever $p(t, f) \neq p(t, f_n)$. Using the same argument with trajectories $t, u \in \mathcal{T}[X]$ and a single objective f, it can be found that $\mathcal{D}$ is also discontinuous in trajectories under the norm topology unless $p(t, f) = p(u, f)$ for all u close to t. Thus most deterministic optimizers, including Newton and quasi-Newton methods as well as simplicial methods and basic generating set search (without a randomized search heuristic), are norm-discontinuous.

It is possible to construct a space of deterministic optimizers in which some deterministic optimizers are continuous. This can be done by starting with the function $p(t, f)$ above, which is a function from $\mathcal{T}[X] \times \mathbb{R}^X$ to X. Call the space consisting of all such functions $\mathcal{DF}$. Then $\mathcal{DF}$ is isomorphic with the set of deterministic optimizers in $\mathcal{MF}$ through the isomorphism $\mathcal{D}[t, f](dx) = \delta_{p(t,f)}(x)$. Deterministic optimizers are continuous in objectives under the given topology for X if $p(t, f_n) \to p(t, f)$ in τ whenever $f_n \to f$. Newton and quasi-Newton methods are continuous in objectives over $\mathcal{DF}$ with this topology, and are also continuous in trajectories on continuously differentiable objectives.

5.3 Evolutionary Algorithms

As discussed in Chapter 4, an evolutionary algorithm can be represented as a convolution of selection, recombination, and variation processes, $\mathcal{E} = \mathcal{S} \star \mathcal{R} \star \mathcal{V}$. Evolutionary algorithms can be continuous or discontinuous, depending on the details of the genetic operators. Mutation operators are independent of objectives and therefore trivially continuous in objectives. Typically, mutation operators are continuous in trajectories as well, as with Bernoulli or Gaussian mutation. Crossover rules are likewise independent of objectives and therefore continuous over objectives. In order to determine when evolutionary algorithms as a whole are continuous, more work is required. Cases where evolutionary algorithms are continuous or discontinuous will be addressed with general theorems in this section. These results can then be used in conjunction with the results of Chapter 7 to conclude when the performance of evolutionary algorithms changes continuously with the objective. That is, these results will make it possible to determine when similarity of objectives permits us to conclude that a particular algorithm will perform similarly on both objectives.

5.3.1 Continuity of Convolution

Since evolutionary algorithms have been formalized as a convolution of optimizers, a study of the continuity of evolutionary algorithms can benefit from discovering whether convolution preserves continuity. The two theorems below demonstrate two distinct cases in which a convolution can be continuous. First, a convolution $\mathcal{A} \star \mathcal{B}$ is continuous if both optimizers are continuous. Second, a convolution may be continuous if the right side is continuous and the left side generates convergent samples. These theorems will be stated for all of $\mathcal{MF}$ and not just $\mathcal{PF}$, and so the concept of *bounded magnitude* must be introduced first.

Definition 5.3.1. *An optimizer* $\mathcal{G} \in \mathcal{MF}$ *is of bounded magnitude if there exists a number* $M < \infty$ *such that* $||\mathcal{G}[t, f]||_{\mathcal{M}} \leq M$ *for all* t, f*.*

An optimizer of bounded magnitude cannot grow without bound on some sequence of objectives or trajectories, which is important because otherwise such a sequence could be used to create a discontinuity during convolution, even when two continuous optimizers are being convolved. Any optimizer in $\mathcal{PF}$ is of bounded magnitude, with $||\mathcal{G}[t, f]||_{\mathcal{M}} \leq 1$, so that this condition is satisfied trivially for the optimizers of interest.

Theorem 5.3.1. *Let* $\mathcal{S}, \mathcal{V} \in \mathcal{MF}$*. Then* $\mathcal{S} \star \mathcal{V}$ *is continuous in objectives (or trajectories) at* t, f *if both* $\mathcal{S}$ *and* $\mathcal{V}$ *are continuous in objectives (or trajectories) at* t, f *and of bounded magnitude.*

Proof. Assume that $\mathcal{S}$ and $\mathcal{V}$ are continuous in both objectives and trajectories. Suppose $||\mathcal{S}[u, g]||_{\mathcal{M}} \leq M < \infty$ and $||\mathcal{V}[u, g]|| \leq M$ for all u, g. Let $f_n \to f$, $t_n \to t$. Let $A \in \mathcal{B}_\tau$. Fix $\epsilon > 0$. Then

$$\left|\mathcal{S} \star \mathcal{V}\left[t_n, f_n\right](A) - \mathcal{S} \star \mathcal{V}\left[t, f\right](A)\right| \tag{5.5}$$

$$= \left|\int_X \mathcal{V}\left[t_n \cup x, f_n\right](A)\mathcal{S}\left[t_n, f_n\right](dx) - \mathcal{V}\left[t \cup x, f\right](A)\mathcal{S}\left[t, f\right](dx)\right| \tag{5.6}$$

$$\leq \left|\int_X \mathcal{V}\left[t_n \cup x, f_n\right](A)\mathcal{S}\left[t_n, f_n\right](dx) - \mathcal{V}\left[t \cup x, f\right](A)\mathcal{S}\left[t_n, f_n\right](dx)\right| \tag{5.7}$$

$$+ \left|\int_X \mathcal{V}\left[t \cup x, f\right](A)\mathcal{S}\left[t_n, f_n\right](dx) - \mathcal{V}\left[t \cup x, f\right](A)\mathcal{S}\left[t, f\right](dx)\right| \tag{5.8}$$

$$\leq \int_X \left|\mathcal{V}\left[t_n \cup x, f_n\right](A) - \mathcal{V}\left[t \cup x, f\right](A)\right| \;\; \left|\mathcal{S}\left[t_n, f_n\right](dx)\right| \tag{5.9}$$

$$+ \int_X \left|\mathcal{V}\left[t \cup x, f\right](A)\right| \;\; \left|\mathcal{S}\left[t_n, f_n\right](dx) - \mathcal{S}\left[t, f\right](dx)\right| \tag{5.10}$$

$$< \frac{\epsilon}{2}\frac{1}{M} \;\; \left|\mathcal{S}\left[t_n, f_n\right]\right|(X) \;\; + \;\; M\left|\mathcal{S}\left[t_n, f_n\right] - \mathcal{S}\left[t, f\right]\right|(X) \tag{5.11}$$

$$< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon. \tag{5.12}$$

To obtain Equation 5.11, use the fact that $\mathcal{V}$ is continuous on the left side and the fact that $\mathcal{V}$ is of bounded magnitude on the right. To obtain Equation 5.12, note that $\mathcal{S}$ is bounded in magnitude by M and that $\mathcal{S}$ is continuous. For Equation 5.11, continuity is sufficient to imply that there exists an N independent of x such that

$$\left|\mathcal{V}\left[t_n \cup x, f_n\right](A) - \mathcal{V}\left[t \cup x, f\right](A)\right| < \frac{\epsilon}{2}\frac{1}{M}$$

for all $n > N$ because $d_\rho(t_n \cup x, t \cup x) = d_\rho(t_n, t)$ for all $x \in X$, using d_ρ from Equation 5.2. This justification can be extended to general topological spaces; the details are not included here.

The proof above holds for jointly continuous $\mathcal{S}$ and $\mathcal{V}$; continuity in either objectives or trajectories separately can be proven by repeating the equations above with $t_n = t$ or $f_n = f$ as needed. $\square$

Theorem 5.3.1 can be applied to evolutionary algorithms to deduce continuity based on the continuity of the selection rules, the crossover rule, and the mutation operator. There is a problem with this approach, however, since most selection and crossover rules are discontinuous in some sense. The following proposition and its corollaries explicitly specify the chain of deductions.

Proposition 5.3.2. *Every crossover rule or mutation operator is continuous in objectives.*

Proof. Let $\mathcal{C}$ be a crossover rule. By the definition of a crossover rule, $\mathcal{C}[t, f] = \mathcal{C}[t, g]$ for all t, f, g, and so it is trivial that $\mathcal{C}$ is continuous in objectives. The same argument holds for mutation operators as well. $\square$

Corollary 5.3.3. *A recombination operator is continuous in objectives at t, f if its selection rules are continuous in objectives at t, f.*

Proof. Let $\mathcal{R}$ be a recombination operator. Then $\mathcal{R} = \triangleleft\mathcal{S}_1 \star (\cdots \star (\triangleleft_n\mathcal{S}_n \star \mathcal{C}))$ for selection operators $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and a crossover rule $\mathcal{C}$. Each component is in $\mathcal{PBO}_K$ and is therefore of bounded magnitude. The crossover rule $\mathcal{C}$ is continuous in objectives, as are the selection rules. By a recursive application of Theorem 5.3.1, $\mathcal{R}$ is continuous in objectives. $\square$

Corollary 5.3.4. *An evolutionary algorithm is continuous in objectives at* t, f *if its selection rules are continuous in objectives at* t, f.

Proof. Let $\mathcal{E}$ be a evolutionary algorithm. Then $\mathcal{E} = \mathcal{S} \star \mathcal{R} \star \mathcal{V}$ where $\mathcal{R}$ is a recombination operator and $\mathcal{V}$ is a mutation operator. Plainly, $\mathcal{V}$ is continuous and of bounded magnitude (since $\mathcal{V} \in \mathcal{PBO}_K$, a subset of $\mathcal{PF}$, which contains only probability measures). $\mathcal{S}$ is continuous by assumption. Furthermore, $\mathcal{R}$ is continuous (and of bounded magnitude) by the previous corollary. Two applications of Theorem 5.3.1 to $\mathcal{S} \star \mathcal{R} \star \mathcal{V}$ complete the proof. □

The previous theorems provide several tools to show that evolutionary algorithms are continuous in objectives if their selection rules are continuous in objectives. A stronger conclusion is possible using a different approach, described next.

5.3.2 Sample Convergence and Continuity

Selection and recombination in evolutionary algorithms typically can only choose from a finite set of points. The parents must come from the prior generation, and there are only finitely many ways that the parents can be recombined. Selection and crossover are often norm-discontinuous for the same reason that deterministic optimizers are norm-discontinuous; convergence of the selected points does not imply norm-convergence in $||\cdot||_{\mathcal{M}}$. In evolutionary algorithms, mutation varies the recombined point, spreading it out so that after mutation, any point in the search space can be generated. The mutation process restores continuity under certain conditions even when the selection and crossover rule are not continuous. These concepts are stated formally in the following definition and theorem.

Definition 5.3.2. *An optimizer* $\mathcal{G} \in \mathcal{MF}$ *is sample convergent in trajectories at* t, f *if*

1. *there is a trajectory* $u_{t,f} \in \mathcal{T}[X]$ *s.t.* $\{y \in u_{t,f}\}$ *has full measure on* $\mathcal{G}[t, f]$,
2. $t_n \to t$ *implies* $\exists u_{t_n,f}$ *as in the prior statement, and* $u_{t_n,f} \to u_{t,f}$, *and*
3. $t_n \to t$ *implies* $\mathcal{G}[t_n, f](\{u^i_{t_n,f}\}) \to \mathcal{G}[t, f](\{u^i_{t,f}\})$ *for all* $1 \leq i \leq |u_{t,f}|$.

If the above statements hold when $t_n \to t$ *is replaced with* $f_n \to f$*, then* $\mathcal{G}$ *is sample convergent in objectives at* t, f*.*

The name *sample convergent* is chosen to reflect the fact that a sample from a sample convergent optimizer converges along a sequence of trajectories or objectives. That is, if $Y_{t,f} \sim \mathcal{G}[t,f]$ for all t, f, then Y_{t_n,f_n} converges in distribution to $Y_{t,f}$ when $t_n, f_n \to t, f$. Sample convergence is not just important for evolutionary algorithms; for example, it can also be used to show when stochastic gradient descent is continuous.

Theorem 5.3.5. *Suppose* $\mathcal{G} \in \mathcal{MF}$*. If* $\mathcal{G}$ *can be written as* $\mathcal{A} \star \mathcal{B}$ *where* $\mathcal{A}$ *and* $\mathcal{B}$ *are both of bounded magnitude,* $\mathcal{A}$ *is sample convergent in objectives (or trajectories) at* t, f*, and* $\mathcal{B}$ *is continuous in objectives (or trajectories) at* t, f*, then* $\mathcal{G}$ *is continuous in objectives (or trajectories) at* t, f*.*

Proof. Without loss of generality, suppose $\mathcal{A}$ is sample convergent in both objectives and trajectories and that $\mathcal{B}$ is continuous in both objectives and trajectories. Fix $\epsilon > 0$ and suppose $\mathcal{A} \leq M$ and $\mathcal{B} \leq M$. The optimizer $\mathcal{G}$ can be written as

$$\mathcal{G}[t,f](A) = \sum_{i=1}^{|u_{t,f}|} \mathcal{A}[t,f](\{u^i_{t,f}\}) \; \mathcal{B}[t \cup u^i_{t,f}, f](A). \tag{5.13}$$

To reduce notation, let $N = |u_{t,f}|$, $p(i,t,f) = \mathcal{A}[t,f](\{u^i_{t,f}\})$, and $\nu_{i,t,f}(A) = \mathcal{B}[t \cup u^i_{t,f}, f](A)$. Then the above can be restated as

$$\mathcal{G}[t,f](A) = \sum_{i=1}^{N} p(i,t,f) \; \nu_{i,t,f}(A). \tag{5.14}$$

Suppose now that $t_n \to t$ and $f_n \to f$. Because $\mathcal{A}$ is sample convergent, it follows that $p(i, t_n, f_n) \to p(i,t,f)$. Also, $\nu_{i,t_n,f_n}(A) \to \nu_{i,t,f}$ since $\mathcal{B}$ is continuous. But then

$$\begin{aligned}
|\mathcal{G}[t,f](A) - \mathcal{G}[t_n,f_n](A)| &\leq \sum_{i=1}^{N} |p(i,t,f)\nu_{i,t,f}(A) - p(i,t_n,f_n)\nu_{i,t_n,f_n}(A)| \\
&\leq \sum_{i=1}^{N} |p(i,t,f)\nu_{i,t,f}(A) - p(i,t,f)\nu_{i,t_n,f_n}(A)| \\
&\quad + \sum_{i=1}^{N} |p(i,t,f)\nu_{i,t_n,f_n}(A) - p(i,t_n,f_n)\nu_{i,t_n,f_n}(A)| \\
&= \sum_{i=1}^{N} |p(i,t,f)|\, |\nu_{i,t,f}(A) - \nu_{i,t_n,f_n}(A)| \\
&\quad + \sum_{i=1}^{N} |\nu_{i,t_n,f_n}(A)|\, |p(i,t,f) - p(i,t_n,f_n)| \\
&\leq M \sum_{i=1}^{N} \left[\frac{\epsilon}{2NM} + \frac{\epsilon}{2NM}\right] \\
&= \epsilon \qquad (5.15)
\end{aligned}$$

where the next to last line follows from the convergence of p and ν mentioned above and from the bounded magnitude of $\mathcal{A}$ and $\mathcal{B}$. Thus $\mathcal{G}$ is continuous in both objectives and trajectories. To show $\mathcal{G}$ is only continuous in objectives or trajectories separately, repeat the above steps with $t_n = t$ or $f_n = f$. □

Corollary 5.3.6. *An evolutionary algorithm is continuous in objectives at t, f if its selection rules are sample convergent in objectives at t, f.*

Proof. It has already been shown that crossover rules and mutation operators are continuous in trajectories. Suppose that $\mathcal{E}$ is an evolutionary algorithms with standard decomposition $\mathcal{E} = \mathcal{S} \star \mathcal{R} \star \mathcal{V}$. Then

$$\mathcal{S} \star \mathcal{R} = \mathcal{S} \star (\triangleleft \mathcal{S}_1 \star (\cdots \star (\triangleleft_n \mathcal{S}_n \star \mathcal{C})))$$

for selection operators $\mathcal{S}_1, \ldots, \mathcal{S}_n$ and a crossover rule $\mathcal{C}$. $\mathcal{C}$ is continuous in objectives, and $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are sample convergent in objectives, as is $\mathcal{S}$. All components are in $\mathcal{PF}$, and thus of bounded magnitude. Recursive application of Theorem 5.3.5 yields that $\mathcal{R}$ is continuous in objectives. Theorem 5.3.1 implies that $\mathcal{E}$ is continuous as well. □

The next theorem shows that masked crossover rules are sample convergent if they have sample convergent selection rules. Since most crossover rules are masked crossover rules, this fact implies that the continuity of most evolutionary algorithms depends on the sample convergence of the selection rule.

Theorem 5.3.7. *The convolution of a selection rule and a recombination operator with a masked crossover rule is sample convergent in objectives (or trajectories) at* t, f *if its selection rules are also sample convergent in objectives (or trajectories) at* t, f.

Proof. Suppose X is a d-dimensional vector space, so that a masked crossover rule can be applied. Let $\mathcal{S}$ be a selection rule that is sample convergent in both trajectories and objectives. Let $\mathcal{R}$ be a recombination operator with a masked crossover rule. Then $\mathcal{S}\star\mathcal{R} = \mathcal{S}\star(\lhd\mathcal{S}_1 \star (\cdots \star (\lhd_{n-1}\mathcal{S}_{n-1} \star \mathcal{C}<\mathbb{P}_M>)))$ for sample convergent selection rules $\mathcal{S}_1, \ldots, \mathcal{S}_{n-1}$ and a masked crossover rule $\mathcal{C}<\mathbb{P}_M>$ of order n. Assume for now that each selection rule is sample convergent in both objectives and trajectories. Let $\mathcal{S}_0 = \mathcal{S}$ to simplify the notation that follows.

For all t, f there is a trajectory $u_{i,t,f}$ for each selection rule $\mathcal{S}_i$ with $i = 0, \ldots, n-1$ such that $\mathcal{S}_i[t,f](\{y \in u_{i,t,f}\}) = 1$. There are n^d possible crossover masks, and each selection rule can only select one of $|u_{i,t,f}|$ points. Thus there are exactly $n^d \prod_i |u_{i,t,f}| < \infty$ points that can result from recombination, and these points may be enumerated within a trajectory $\tilde{u}_{t,f}$, where the order of enumeration is independent of t and f. To be specific, for each position k in $\tilde{u}_{t,f}$ there is a crossover mask m^k and an index to a selected parent $p_{i,k}$ for each selection rule i such that m^k and $(p_{i,k})_{i=1}^n$ depend solely on the position k and not on t, f. Recalling Equation 4.15, $\tilde{u}^k_{t,f} = \sum_{i=1}^n m^k \otimes_i u^{p_{i,k}}_{i,t,f}$, and $\mathcal{S} \star \mathcal{R}[t,f](\{y \in \tilde{u}_{t,f}\}) = 1$.

Suppose $t_n \to t$ and $f_n \to f$. Then $u_{i,t_n,f_n} \to u_{i,t,f}$ for each selection rule $\mathcal{S}_i$. Let $x = \tilde{u}^k_{t,f}$, the k^{th} element of the trajectory $\tilde{u}_{t,f}$. Then x is generated from a particular crossover mask m determined by the position k. Suppose m has the value j in the ℓ^{th} component, i.e. $m_\ell = j$. Then $x_\ell = \left(u^k_{j,t,f}\right)_\ell$. Let $x^n = \tilde{u}^k_{t_n,f_n}$. Then because the enumeration order was fixed, $x^n_\ell = \left(u^k_{j,t_n,f_n}\right)_\ell$. Since $u_{j,t_n,f_n} \to u_{j,t,f}$, it follows that $x^n_\ell \to x_\ell$. But k, j, and ℓ were arbitrary, so it follows that $\tilde{u}_{t_n,f_n} \to \tilde{u}_{t,f}$.

Again, suppose $t_n \to t$ and $f_n \to f$. Let m be the crossover mask for $u^k_{t,f}$, and let $y_i = u^{p_{i,k}}_{i,t,f}$ be the point selected on t, f by the i^{th} selection rule at the k^{th} position in the enumeration. Observe that

$$\mathcal{S} \star \mathcal{R}[t, f](\{\tilde{u}^k_{t,f}\}) = \mathbb{P}_M(m) \prod_i \mathcal{S}_i[t, f](\{y_i\}). \tag{5.16}$$

Let $y^n_i = u^{p_{i,k}}_{i,t_n,f_n}$ be the point selected on t_n, f_n by the i^{th} selection rule at the k^{th} position in the enumeration and note that (1) $y^n_i \to y_i$, (2) $\tilde{u}^k_{t_n,f_n} \to \tilde{u}^k_{t,f}$, and (3) the particular mask m is a function of the position k independent of t, f. Since $\mathbb{P}_M(m)$ is independent of t, f and $\mathcal{S}_i[t_n, f_n](\{y^n_i\}) \to \mathcal{S}_i[t, f](\{y_i\})$ for all i, it follows that $\mathcal{S} \star \mathcal{R}[t_n, f_n](\{\tilde{u}^k_{t_n,f_n}\}) \to \mathcal{S} \star \mathcal{R}[t, f](\{\tilde{u}^k_{t,f}\})$. Therefore $\mathcal{S} \star \mathcal{R}$ is pointwise convergent. To show that $\mathcal{S} \star \mathcal{R}$ is only convergent in either trajectories or objectives, repeat the above with $f_n = f$ or $t_n = t$. □

Corollary 5.3.8. *An evolutionary algorithm with a masked crossover rule is continuous in trajectories (or objectives) at t, f if its mutation operator is continuous in trajectories (or objectives) at t, f and its selection rules are sample convergent in trajectories (or objectives) at t, f.*

Proof. Let $\mathcal{E} = \mathcal{S} \star \mathcal{R} \star \mathcal{V}$ be an evolutionary algorithm with its standard decomposition. Then $\mathcal{S} \star \mathcal{R}$ is sample convergent by Theorem 5.3.7, and so $\mathcal{E}$ is continuous by Theorem 5.3.5. □

As a final piece of the puzzle, proportional selection is sample convergent on $\overline{C[X]}$ under certain conditions. [2] Recall that $\mathcal{PS}\langle g \rangle$ from Equation 4.10 is generalized proportional selection with a modulating function g, so that each point y from the prior population $H(t)^{-1}$ is selected proportionately to $g \circ f$, where f is the objective function. The theorem below and its corollary proves that the simple genetic algorithm.

Theorem 5.3.9. *Proportional selection with modulating function g is sample convergent on all trajectories and all objectives in $\overline{C[X]}$ if and only if g is continuous on the image of f.*

---

[2] $\overline{C[X]} \subseteq \mathbb{R}^X$ consists of all continuous real functions and their pointwise limits, including functions with jump discontinuities or point discontinuities.

Proof. To make the proof simpler, use unnormalized proportional selection,

$$\mathcal{UPS}{<}g{>}[t,f]\,(B) = \sum_{k=1}^{K} g(t, f\left(H(t)^{-1,k}\right))1_B(x), \tag{5.17}$$

noting that $H(t)^{-1}$ is a sequence that may repeat points.

Suppose $t_n \to t$ and $f_n \to f$. Without loss of generality, suppose f_n is continuous, as we may, since continuous functions are dense in $\overline{C[X]}$. Clearly, the set $P_{t,f} = \{y \in H(t)^{-1}\}$ has full measure on $\mathcal{UPS}[t,f]$ for all t, f, and $H(t_n)^{-1} \to H(t)^{-1}$ in X^K (or in $\mathcal{T}[X]$). It remains to show that

$$\mathcal{UPS}[t_n, f_n](\{H(t_n)^{-1,k}\}) \to \mathcal{UPS}[t,f](\{H(t)^{-1,k}\})$$

for all k.

$$\begin{aligned} \left|\mathcal{UPS}[t_n, f_n](\{H(t_n)^{-1,k}\})\right. &- \left.\mathcal{UPS}[t,f](\{H(t)^{-1,k}\})\right| \\ &= \left|g(t, f_n\left(H(t_n)^{-1,k}\right)) - g(t, f\left(H(t)^{-1,k}\right))\right| \end{aligned} \tag{5.18}$$

Now f_n is continuous and $f_n \to f$, so for any $\epsilon > 0$,

$$\begin{aligned} \left|f_n\left(H(t_n)^{-1,k}\right) - f\left(H(t)^{-1,k}\right)\right| &\leq \left|f_n\left(H(t_n)^{-1,k}\right) - f_n\left(H(t)^{-1,k}\right)\right| \\ &\quad + \left|f_n\left(H(t)^{-1,k}\right) - f\left(H(t)^{-1,k}\right)\right| \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} < \epsilon. \end{aligned} \tag{5.19}$$

Since g is continuous, the desired conclusion follows by normalizing $\mathcal{UPS}$. □

Corollary 5.3.10. *The simple genetic algorithm $\mathcal{SGA}$ of Equation 4.20 is jointly continuous in trajectories and objectives.*

Proof. Recall that $\mathcal{SGA}{<}p{>}= (\mathcal{PS}{<}|x|{>} \star ((\lhd\mathcal{PS}{<}|x|{>}) \star \mathcal{SC})) \star \mathcal{B}{<}p{>}$, where the objective is assumed to be negative (for minimization). The search space is $\{0,1\}^d$ with the discrete topology (i.e. all sets are open), and therefore $\overline{C[\{0,1\}^d]} = \mathbb{R}^X$. The function $g(x) = |x|$ is continuous, and so $\mathcal{PS}{<}|x|{>}$ is sample convergent everywhere by Theorem 5.3.9. The Bernoulli mutation operator $\mathcal{B}{<}p{>}$ is jointly continuous. Single-point crossover $\mathcal{SC}$ is a masked crossover rule, so Corollary 5.3.8 implies that $\mathcal{SGA}$ is jointly continuous everywhere. □

Genetic algorithms in any space are jointly continuous on all trajectories and objectives in $\overline{C[X]}$ when they use masked crossover and proportional selection with a continuous modulating function. For example, a real-coded genetic algorithm with proportional selection, uniform crossover, and Gaussian mutation is continuous in this way.

Proportional selection (also called roulette wheel selection) is no longer commonly used as a selection rule because of its sensitivity to numeric values, its requirement of a negative fitness function, and its inability to prefer more refined solutions near an optimum. It has been replaced by tournament selection and ranking selection. Whereas roulette wheel selection makes a genetic algorithm continuous, tournament and ranking selection are discontinuous at some points. The following sections identify these discontinuities, leading up to a full characterization of when exactly the more commonly used selection rules are continuous.

5.3.3 Sample Divergence and Discontinuity

In the previous section, sample convergence was used to show that many genetic algorithms are continuous on a large set of objectives. In this section, similar proofs will be used to demonstrate a converse result, that selection rules whose samples diverge are a source of discontinuities in the optimizer. The concept of sample divergence is defined next, followed by the converse of Theorem 5.3.5.

Definition 5.3.3. *An optimizer $\mathcal{G} \in \mathcal{MF}$ is sample divergent in trajectories at t, f if*

1. *there is a trajectory $u_{t,f} \in \mathcal{T}[X]$ s.t. $\{y \in u_{t,f}\}$ has full measure on $\mathcal{G}[t, f]$,*
2. *$t_n \to t$ implies $\exists u_{t_n,f}$ as in the prior statement, and $u_{t_n,f} \to u_{t,f}$,*
3. *$t_n \to t$ implies $\mathcal{G}[t_n, f](\{u^i_{t_n,f}\}) \nrightarrow \mathcal{G}[t, f](\{u^i_{t,f}\})$ for some $1 \leq i \leq |u_{t,f}|$.*

If the above statements hold when $t_n \to t$ is replaced with $f_n \to f$, then $\mathcal{G}$ is sample divergent in objectives at t, f.

Theorem 5.3.11. *Suppose $\mathcal{G} \in \mathcal{MF}$. If $\mathcal{G}$ can be written as $\mathcal{A} \star \mathcal{B}$ where $\mathcal{A}$ and $\mathcal{B}$ are both of bounded magnitude, $\mathcal{A}$ is sample divergent in objectives (or trajectories) at t, f, and $\mathcal{B}$ is continuous in objectives (or trajectories) at t, f, then $\mathcal{G}$ is discontinuous in objectives (or trajectories) at t, f.*

Proof. Without loss of generality, assume that $\mathcal{A}$ is sample convergent in both objectives and trajectories at t, f and that $\mathcal{B}$ is continuous in both objectives and trajectories at t, f. Suppose $t_n \to t$ and $f_n \to f$. Fix $A \in \mathcal{B}_\tau$. Adopt notation for $\mathcal{G}$ as in Equation 5.14. Then there is some i such that $p(i, t_n, f_n) \nrightarrow p(i, t, f)$, i.e. $|p(i, t_n, f_n) - p(i, t, f)| = c_1 > 0$. Also, let $c_2 = \sum_{i=1}^N \nu_{i,t,f}(A) > 0$.

$$\begin{aligned}
||\mathcal{G}[t, f] - \mathcal{G}[t_n, f_n]||_\mathcal{M} &\geq |\mathcal{G}[t, f](A) - \mathcal{G}[t_n, f_n](A)| \\
&= \left| \sum_{i=1}^N p(i, t, f)\nu_{i,t,f}(A) - p(i, t_n, f_n)\nu_{i,t_n,f_n}(A) \right| \\
&= \left| \sum_{i=1}^N \left[p(i, t, f) - p(i, t_n, f_n)\right] \nu_{i,t,f}(A) \right. \\
&\qquad \left. + \sum_{i=1}^N p(i, t_n, f_n) \left[\nu_{i,t,f}(A) - \nu_{i,t_n,f_n}(A)\right] \right| \\
&\geq \left| c_1 c_2 - \frac{c_1 c_2}{2} \right| = \frac{c_1 c_2}{2} > 0, \qquad (5.20)
\end{aligned}$$

where the factor $\frac{c_1 c_2}{2}$ is introduced because of the continuity of ν. Thus $||\mathcal{G}[t, f] - \mathcal{G}[t_n, f_n]||_\mathcal{M}$ does not converge for $\epsilon < c_1 c_2$, and $\mathcal{G}$ is discontinuous at t, f. □

Theorem 5.3.7 stated that a masked crossover rule preserves sample convergence from its selection rules. An analogue to this theorem is true; masked crossover also preserves sample divergence. The following Theorem and Corollary can be proven in a similar way to Theorem 5.3.7 and its corollaries, and so the proofs are omitted.

Theorem 5.3.12. *A recombination operator with a masked crossover rule is sample divergent in objectives (or trajectories) at t, f if all of its selections rules are sample divergent in objectives (or trajectories) at t, f.*

Corollary 5.3.13. *An evolutionary algorithm with a masked crossover rule is discontinuous in objectives (or trajectories) at* t, f *if all of its selection rules are sample divergent in objectives (or trajectories) at* t, f *and its mutation operator is continuous in objectives (or trajectories).*

5.3.4 Discontinuities of Specific Selection Rules

Theorem 5.3.9 showed that generalized proportional selection is sample convergent where the modulating function is continuous on the image of the objective function. The next result shows the opposite. Proportional selection is sample divergent when the composition of the modulating function and the objective is discontinuous.

Theorem 5.3.14. *Proportional selection with modulating function* g *is sample divergent in objectives and trajectories at* t, f *whenever its modulating function* g *is discontinuous on the image of* f *at the evaluation point* $H(t)^{-1,k(t)}$ *in the prior population of* t.

Proof. Let $k = k(t)$, and let $x_k = H(t)^{-1,k}$ be a discontinuity point of $g(t, f(\cdot))$ in accordance with the assumptions. Suppose $t_n \to t$ and $f_n \to f$ but $g(t_n, f_n(x_k)) \not\to g(t, f(x_k))$. As in the proof of Theorem 5.3.9, use unnormalized proportional selection. Also as in that proof, $\mathcal{PS}\!<\!g\!>$ meets the basic requirements of sample divergence (or convergence), i.e. $u_{t,f} = H(t)^{-1}$ and $P_{t,f} = \{y \in H(t)^{-1}\}$ has full measure. Let $x_k^n = H(t_n)^{-1,k}$. The goal is now to demonstrate that $\mathcal{PS}[t_n, f_n](\{x_k^n\}) \not\to \mathcal{PS}[t, f](\{x_k\})$. It is not difficult to do so, because

$$\begin{aligned}
|\mathcal{PS}[t_n, f_n](\{x_k^n\}) &- \mathcal{PS}[t, f](\{x_k\})| \\
&= |g(t_n, f_n\left(x_k^n\right)) - g(t, f\left(x_k\right))| \\
&= |[g(t_n, f_n\left(x_k^n\right)) - g(t_n, f_n\left(x_k\right))] \\
&\quad + [g(t_n, f_n\left(x_k\right)) - g(t, f\left(x_k\right))]| \,, \qquad (5.21)
\end{aligned}$$

and whether or not $g(t_n, f_n(\cdot))$ is continuous at x_k, this sum can be bounded below by a constant greater than zero. Therefore $\mathcal{PS}$ is discontinuous at t, f. □

The proof of Theorem 5.3.14 can be leveraged to conclude that tournament selection and ranking selection are also sample divergent on the majority of objectives on certain trajectories. The following definition will make explicit the trajectories on which this discontinuity occurs.

Definition 5.3.4. *A trajectory $t \in \mathcal{T}[X]$ is of ambivalent fitness at degree K on an objective f if there exist points $x, y \in H(t)^{-1}$ for population size K with $x \neq y$ but $f(x) = f(y)$. Otherwise, the t is of unambivalent fitness at degree K on f. The trajectory t is ambivalent at full degree if $K = |t|$; the degree may be omitted if clear from the context.*

Notice that a monotonic objective can never produce a trajectory of ambivalent fitness.

Theorem 5.3.15. *Tournament selection (Equation 4.11) and ranking selection (Equation 4.13) are both sample divergent in objectives at every objective on trajectories of ambivalent fitness at the degree of the selection rule.*

Proof. Let $R(y, f, P)$ be the ranking function of Section 4.2.3. Define unnormalized tournament selection by $\mathcal{UTS}\left<q\right>[t, f](\{x\}) = h_1(x)$, where

$$h_1(x) = (1-q)^{R(x,f,H(t)^{-1})}. \tag{5.22}$$

Similarly, define unnormalized ranking selection by $\mathcal{URS}\left<q\right>[t, f]\left(\{x\}\right) = h_2(x)$, where

$$h_2(x) = r_q^{t,f}(x). \tag{5.23}$$

In either case, the functions h_1 and h_2 can be substituted for $g(t, f(\cdot))$ verbatim in the proof of Theorem 5.3.14 to obtain that tournament selection and ranking selection are sample divergent at the discontinuities of h_1 and h_2. Now h_1 and h_2 are continuous functions of $R(x, f, H(t)^{-1})$, and thus their discontinuities are exactly the discontinuities of R.

Let f be any non-monotonic objective and let t be a trajectory of ambivalent fitness on f at the degree of the selection rule, so that there are two points y and z in $H(t)^{-1}$ with $y \neq z$ and $f(y) = f(z)$. Next, construct f_n so that $f_n(z) = f(z) + \frac{1}{n}$ and $f_n(x) = f(x)$ for all $x \neq z$. Then $f_n \to f$, and

$$R(z, f_n, H(t)^{-1}) - R(y, f_n, H(t)^{-1}) > 0$$

is a positive constant independent of n, i.e. y is ranked higher than z, and thus has a lower index in the ranked population. But according to the disambiguation rule in Section 4.2.3,

$$R(z, f, H(t)^{-1}) - R(y, f, H(t)^{-1}) < 0,$$

that is, y is ranked lower than z at the limit and has a higher index in the population. Therefore R is discontinuous in objectives at t, f, and by consequence tournament and ranking selection are discontinuous in objectives at t, f as well. If the tie-breaking procedure is reversed, the proof still holds by using $f_n(z) = f(z) - \frac{1}{n}$ instead. $\square$

These discontinuities are not as serious as they appear at first. In fact, both tournament selection and ranking selection are sample convergent in objectives on trajectories that are not of ambivalent fitness. On most objective functions, these optimizers do not produce trajectories of ambivalent fitness. In fact, such trajectories will have measure zero unless the objective function has a plateau. Even on functions with many small plateaus, trajectories of ambivalent fitness will rarely be encountered.

Theorem 5.3.16. *Tournament selection and ranking selection are both sample convergent on objectives in* $\overline{C[X]}$ *at trajectories that are of unambivalent fitness.*

Proof. As in the proof of Theorem 5.3.15, tournament and ranking selection are sample divergent at exactly the points where R is discontinuous. Let $f \in \overline{C[X]}$, and let t be a trajectory that is of unambivalent fitness on f at the degree of the selection rule. Assume $f_n \to f$. Then there is an n such that $R(x, f_n, H(t)^{-1}) = R(x, f, H(t)^{-1})$ since the population size K is finite, and any finite set of points in $\mathbb{R}$ can be separated by disjoint open sets. But then R is continuous on f at t, and therefore tournament and ranking selection are sample convergent by a repetition of the proof of Theorem 5.3.9 with h_1 and h_2 from the proof of Theorem 5.3.15 replacing $g(t, f(\cdot))$. $\square$

The previous two proofs determine when tournament and ranking selection are sample convergent or divergent in objectives. They depended on the fact that fitness ranking is inherently discontinuous on trajectories of ambivalent fitness.

An analysis of truncation selection will complete this survey of continuity in evolutionary algorithms. Truncation selection, used by evolution strategies and estimation of distribution algorithms, also depends indirectly on the rank. A *truncation selection rule* in $\mathcal{PBO}_K$ places probability one on the best T members of the last population, with $1 \leq T < K$. In Section 4.2.4, evolution strategy selection was defined as $\mathcal{ESS}$ and $\mathcal{ESS}_+$. Both of these selection rules are truncation selection rules. All truncation selection rules are sample divergent in objectives on all monotonic objectives due to the discontinuity in the ranking function that was exploited in the proof of Theorem 5.3.15. The following proposition and its corollary can be proved using the same strategy as for Theorem 5.3.15.

Proposition 5.3.17. *A truncation selection rule is sample divergent in objectives on trajectories of ambivalent fitness, and is sample convergent on objectives in $\overline{C[X]}$ at trajectories of unambivalent fitness.*

Corollary 5.3.18. *Evolution strategies in $\mathbb{R}^d$ with intermediate or dominant crossover and Gaussian mutation are continuous in objectives at t, f if $f \in \overline{C[X]}$ and t is of unambivalent fitness on f; they are discontinuous in objectives on trajectories of ambivalent fitness.*

The principles from the theorems above are not restricted to evolutionary algorithms. Sample convergence is an important and useful concept that can be used to demonstrate the continuity or discontinuity of quasi-evolutionary algorithms and even stochastic gradient descent, as is done in the next two sections.

5.4 Quasi-Evolutionary Algorithms

Quasi-evolutionary algorithms can be described in terms of selection, crossover, and mutation operators, but they typically use much more complex crossover mechanisms. The techniques for determining the continuity of evolutionary algorithms were described in at a general level in Section 5.3 so that the same techniques can be applied to demonstrate the discontinuity of the most popular quasi-evolutionary algorithms with respect to objectives. This section develops this result for the parameterized model-building methods and the class of locally improving optimizers, which includes differential evolution.

5.4.1 Parameterized Methods

In Chapter 2, the class of parameterized quasi-evolutionary methods was introduced. These methods include estimation of distribution algorithms (EDAs) and natural evolution strategies (NES), which subsumes Correlated Matrix Adaption (CMA-ES). The most popular methods in this class are discontinuous in objectives on trajectories of ambivalent fitness, but are mostly continuous elsewhere. When they arise, the discontinuities are due to the use of truncation selection.

An EDA with truncation selection can be represented as the convolution of a truncation selection rule and a model sampler. The model sampling procedure typically does not depend on the objective, but only on the selected members of the population. For this reason, the model sampler is continuous in objectives, and so the EDA can be shown to be continuous or discontinuous in objectives by applying Theorem 5.3.5 or Theorem 5.3.11, depending on whether the trajectory in question is of ambivalent fitness. The same logic also holds for Natural Evolution Strategies, including CMA-ES.

The proofs in Section 5.3 were developed abstractly. A proof that is specific to a known algorithm may be help to make the meaning of these results more concrete. With this goal in mind, this section shows directly that the Bayesian Optimization Algorithm (BOA), a popular EDA, is discontinuous in objectives. To this end, let $\mathcal{BOA}\left\langle T,K\right\rangle \in \mathcal{PBO}_K$ represent BOA with a truncation size of T. $\mathcal{BOA}\left\langle T,K\right\rangle$ builds a directed graphical model from the best T individuals out of a population of size K by employing a greedy hill-climbing search through graph structures using the K2 metric [89, 154–156]. For this example, a binary space is assumed, $X = \{0,1\}^d$.

Proposition 5.4.1. *$\mathcal{BOA}\left\langle T,K\right\rangle$ is discontinuous in objectives on $X = \{0,1\}^d$ when $1 < T < K$.*

Proof. Let $f_n(x) = \frac{1}{n}\delta_1(x_0)$ with δ as the Kronecker delta and x_0 the first bit of x, and let $f(x) = 0$. Then $f_n \to f$. Note that for any $x \in X$, $f_n(x) = \frac{1}{n}$ if $x_0 = 1$ and $f_n(x) = 0$ if $x_0 = 0$. But $f(x) = 0$ for all x. Let $A \equiv \{x \in X : x_0 = 1\}$ and let $B \equiv \{x \in X : x_0 = 0\}$. Let t be a trajectory of length K, and let t have exactly T elements in A and $K - T$ elements in B. The T elements in A are more optimal on f_n than the elements in B because BOA builds a model out of the best T elements. Then for some fixed $\epsilon > 0$ determined by the smoothing procedure for model estimation,

$$\mathcal{BOA}\left\langle T, K\right\rangle [t, f_n]\,(A) = 1 - \epsilon. \tag{5.24}$$

But on f all elements in t are equally optimal. Now assume without loss of generality that truncation selection prefers elements in B over elements in A, so that the model constructed by BOA for f should produce elements from B with approximate probability $\frac{K-T}{K}$, i.e. $\mathcal{BOA}\left\langle T, K\right\rangle [t, f](A) \approx \frac{K-T}{K} \neq 1 - \epsilon$. $\square$

This proof shows how discontinuities can appear in algorithms such as BOA. Notice that the discontinuity in the proof above exists because the objective function was chosen to be identically zero, which implies that all trajectories are of ambivalent fitness. Like other EDAs, BOA is continuous in objectives on trajectories of unambivalent fitness.

5.4.2 Differential Evolution and Locally Improving Optimizers

In Sections 5.3 and 5.4.1, optimizers that depend on the objective rank of evaluation points have been repeatedly shown to be discontinuous in objectives exactly on trajectories of ambivalent fitness. Differential evolution also depends on the objective rank but in a different way from the previously analyzed optimizers, and thus a different type of continuity proof is required.

To generalize the result, consider population-based optimizers with population size K that depend only on the local best solutions for each member of the population. Given a trajectory t, define the best running population by $\text{best}\,(t, f) \in X^K$ so that t is treated as K separate trajectories, and $\text{best}\,(t, f)$ stores the best individual along each of these trajectories. Formally, $\text{best}(t, f)_k = \text{argmax}_{\left\{H(t)^{n,k} : 1 \leq n \leq |H(t)|\right\}} f\left(H(t)^{n,k}\right)$, where $H(t)^{n,k}$ is the k^{th}

individual in the n^{th} population derived from the trajectory t. To resolve ambiguities, let $\text{best}(t, f)$ take on the value corresponding to the largest n. Such optimizers will be termed *locally improving*.

Definition 5.4.1. *An optimizer* $\mathcal{G} \in \mathcal{MF}$ *is locally improving if* $\mathcal{G}[t_1, f] = \mathcal{G}[t_2, f]$ *if and only if* $\text{best}(t_1, f) = \text{best}(t_2, f)$*.*

As an aside, the space of locally improving optimizers forms a vector subspace of $\mathcal{MF}_{\text{tr}}$, because the locally improving property is trajectory-restricted and is preserved by vector operations.

In all but the simplest search domains, locally improving optimizers are continuous in objectives on trajectories satisfying an analogue of the unambivalent fitness requirement of the previous sections. The following definition extends the definition of ambivalent fitness to account for the structure of locally improving optimizers.

Definition 5.4.2. *A trajectory* $t \in \mathcal{T}[X]$ *is of componentwise ambivalent fitness on an objective* f *at degree* K *if for some* k *with* $1 \leq k \leq K$*, there exist* m, n *such that (1)* $H(t)^{m,k} \neq H(t)^{n,k}$*, (2)* $f\left(H(t)^{m,k}\right) = f\left(H(t)^{n,k}\right)$*, and (3)* $f\left(H(t)^{n,k}\right) \leq f\left(H(t)^{i,k}\right)$ *for all* i*. Otherwise,* t *is of componentwise unambivalent fitness on* f *at degree* K*.*

The main concept is the same as in Theorem 5.3.15. Any objective function can be modified to add a mode (or even a plateau) of arbitrary size along trajectories of ambivalent fitness, and when this is done, the vector $\text{best}(t, f)$ changes discontinuously with the objective function as the added mode becomes arbitrarily small. The proof is given next.

Theorem 5.4.2. *Every locally improving optimizer in* $\mathcal{MF}_{\text{ir}}[X]$ *is continuous in objective functions at* t, f *if and only if* t *is of componentwise unambivalent fitness on* f *at full degree.*

Proof. Let $\mathcal{G} \in \mathcal{MF}_{\text{tr}}[X]$ be locally improving with population size K. First, let t be of componentwise ambivalent fitness on f at the appropriate degree. Let k be the component of the vector $\text{best}(t, f)$ along which ambivalent fitness occurs, and let y, z be the two distinct points along the k^{th} slot of t with $y \neq z$

and $f(y) = f(z)$. Let $f_n(x) = f(x)$ for all $x \neq z$. Let $f_n(z) = f(z) + \frac{1}{n}$. Plainly, $f_n \to f$.

Then $best(t, f)_k = z$ but $best(t, f_n)_k = y$ because of the requirement that y and z are the best points in t and because of the disambiguation rule adopted for interpreting best. Therefore $\mathcal{G}[t, f] \neq \mathcal{G}[t, f_n]$. In fact, because $\text{best}(t, f_n)_k$ is the same regardless of n, there is a constant $\epsilon > 0$ such that $||\mathcal{G}[t, f_n] - \mathcal{G}[t, f]||_{\mathcal{M}} = \epsilon$ independent of n, and therefore $\mathcal{G}$ is discontinuous in objectives at t, f. If the disambiguation rule were reversed, then reversing the order of the two populations in t would carry the proof.

If, on the other hand, t is of componentwise unambivalent fitness, then for each population slot k, the fitness of the points along the trajectory t at the k^{th} slot can be separated by disjoint open sets, one for each point. Therefore for n large, $\text{best}(t, f) = \text{best}(t, f_n)$. Therefore $\mathcal{G}[t, f] = \mathcal{G}[t, f_n]$, and $\mathcal{G}$ is continuous in objectives at t, f. $\square$

This theorem makes it clear exactly when a locally improving optimizer is continuous in objectives. These facts will be employed in the next two chapters to conclude that the performance of locally improving optimizers is continuous. That is, these optimizers perform similarly on similar objectives.

5.5 Stochastic Gradient Descent

The concept of sample convergence is not only useful for assessing the continuity of evolutionary and quasi-evolutionary methods. Returning briefly to quasi-Newton methods, a sample convergence argument can be used to conclude that stochastic gradient descent is continuous in objectives on all continuously differentiable objectives.

Stochastic gradient descent in $\mathbb{R}^d$ commonly adds a Gaussian noise factor to the gradient. Let $g(x, f, n)$ be the gradient update of x on an objective f after n steps,

$$g(x, f, n) = x - \eta_n f'(x), \tag{5.25}$$

where η_n is a decreasing learning rate. Define stochastic gradient descent by

$$\mathcal{SG}\langle\sigma\rangle[t, f] = \mathcal{N}\left(g(t^{-1}, f, |t|), \sigma^2\right) \tag{5.26}$$

where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. Then $\mathcal{S}\mathcal{G}$ may be rewritten as a convolution,

$$\mathcal{S}\mathcal{G}\left<\sigma\right> = \mathcal{G} \star \mathcal{N}\left<\sigma\right>, \tag{5.27}$$

where $\mathcal{G}$ is deterministic gradient descent, $\mathcal{G}[t,f](\{g(t^{-1}, f, |t|)\}) = 1$, and $\mathcal{N}\left<\sigma\right>$ is the Gaussian mutation operator of Chapter 4.

Because f is continuously differentiable, $\mathcal{G}$ is sample convergent in objectives on t, f. $\mathcal{N}\left<\sigma\right>$ is trivially continuous in objectives. By Theorem 5.3.5, $\mathcal{S}\mathcal{G}$ is continuous in objectives at t, f whenever f is continuously differentiable.

5.6 Conclusion

This chapter has provided tools to assess the continuity of various optimizers with respect to both objectives and trajectories. Continuity is an important analytical tool because, as the next two chapters will show, continuous optimizers perform similarly on similar objectives.

The next chapter addresses what happens when the optimizer is run for several steps on a particular objective. In this context, continuity in objectives is more important than continuity in trajectories, since the optimizer controls the trajectory but not the objective. Because the continuity of optimizers is important for analyzing optimizer performance, this chapter has reviewed the continuity of a variety of optimizers discussed in previous chapters.

Deterministic optimizers have been shown to be discontinuous everywhere in objectives, although because of their singular nature, deterministic optimizers will still have continuous performance if they are sample convergent.

Evolutionary and quasi-evolutionary methods are continuous in objectives on trajectories of unambivalent fitness. With the exception of locally improving optimizers such as differential evolution, continuity can only be expected of these optimizers for objectives that are not too chaotic, i.e., that reside in $\overline{C[X]}$. In the next chapter, trajectories of ambivalent fitness will be shown to have measure zero when the optimizers are run on a sufficiently large search space, so that these optimizers are continuous almost everywhere with respect to themselves on $\overline{C[X]}$.

The concept of sample convergence or divergence has proven to be theoretically important for stochastic optimizers in general, even though it only applies to singular optimizers. This importance derives from the fact that most popular stochastic optimizers can be decomposed into a convolution of a singular optimizer and a nonsingular one, as was done for stochastic gradient descent.

A final question about continuity is the following: When does continuity of an optimizer imply continuity on the stochastic process generated by that optimizer? This question is addressed in the next chapter, along with an analysis of the long-running behavior of optimizers in general.

Chapter 6

The Optimization Process

In the previous chapter, the continuity of optimizers was explored in terms of how the distribution over the next evaluation point changes when the objective or the trajectory changes. However, optimizers are typically run by calling the same optimizer successively to generate a sequence of evaluation points. In order to analyze optimizer performance in the next chapter, it will be necessary to analyze the random process that is generated by running the optimizer on a fixed objective function. This *optimization process* was briefly introduced in Section 3.2.3. In this chapter, it will be analyzed in detail, paving the way for an analysis of optimizer performance in Chapter 7 and an extension of the No Free Lunch theorems to infinite-dimensional spaces in Chapter 9.

6.1 Construction of the Optimization Process

When a stochastic optimizer $\mathcal{G} \in \mathcal{PF}$ is run on a particular objective f, it is initialized with the empty trajectory, and $\mathcal{G}[\emptyset, f]$ is sampled to obtain a random evaluation point Z_1. This point is added to the trajectory, and $\mathcal{G}[(Z_1), f]$ is sampled to get Z_n. The process continues iteratively, so that $Z_{n+1} \sim \mathcal{G}[(Z_m)_{m=1}^{n}, f]$ for each n. In this way, an infinite random process $Z = (Z_n)_{n=1}^{\infty}$ can be generated. It is not immediately obvious that this infinite process exists or is well-defined. The goal of this section is to construct an infinite random process whose finite-dimensional distributions correspond to the joint distributions over a subset of the Z_n. The process generated in this way is termed the *optimization process* of an optimizer $\mathcal{G}$ on an objective f.

The construction of the optimization process is performed using the Kolmogorov Extension Theorem [105, 112]. This procedure may seem overkill at first since $X^{\mathbb{N}}$ is a countable space, but this same method will be used again later to construct function priors over the potentially uncountable space of objective functions in Chapter 9.

First, the optimization process is an infinite sequence lying in the space $X^{\mathbb{N}}$. A suitable σ-algebra for $X^{\mathbb{N}}$ is thus required; it can be built from *cylinder sets* on X. An n-dimensional cylinder set on $X^{\mathbb{N}}$ is a set of the form

$$A = \left\{ z \in X^{\mathbb{N}} : z_{k_i} \in A_i, 1 \leq i \leq n \right\} \tag{6.1}$$

for an index set $k = (k_1, \ldots, k_n)$ and some $A_i \in \mathcal{B}_\tau$ for $i = 1, \ldots, n$. That is, an n-dimensional cylinder set restricts the values taken on by an infinite sequence at exactly n components. As in [105], let $\mathcal{C}$ be the field containing all cylinder sets on $X^{\mathbb{N}}$, and denote by $\mathcal{B}[X^{\mathbb{N}}]$ the smallest σ-algebra containing $\mathcal{C}$. $\mathcal{B}[X^{\mathbb{N}}]$ is sufficient to support the optimization process.

It remains to construct a probability measure for the optimization process on an optimizer $\mathcal{G} \in \mathcal{PF}$ and a given objective f. Such a measure can be created by patching together a consistent family of finite-dimensional distributions. The following definitions are taken from Karatzas and Shreve [105]:

Definition 6.1.1. *Let T be the set of finite sequences $\tilde{k} = (k_1, \ldots, k_n)$ of distinct nonnegative integers, where the length n of the sequence ranges over the positive integers. Suppose that for each $\tilde{k}$ of length n there is a probability measure $\mathbb{Q}_{\tilde{k}}$ on $(X^n, \mathcal{B}_{\tau^n})$, where τ^n is the product topology on τ. The collection $\{\mathbb{Q}_{\tilde{k}}\}_{\tilde{k} \in T}$ is a family of finite-dimensional distributions.*

The family $\{\mathbb{Q}_{\tilde{k}}\}$ is said to be consistent if it satisfies the following two conditions:

(1) if $\tilde{\ell} = (k_{i_1}, \ldots, k_{i_n})$ is a permutation of $\tilde{k} = (k_1, \ldots, k_n)$, then for any $A_i \in \mathcal{B}_\tau$,

$$\mathbb{Q}_{\tilde{k}}(A_1 \times \cdots \times A_n) = \mathbb{Q}_{\tilde{\ell}}(A_{i_1} \times \cdots \times A_{i_n}). \tag{6.2}$$

(2) if $\tilde{k} = (k_1, \ldots, k_n)$ with $n \geq 1$, $\tilde{\ell} = (k_1, \ldots, k_{n-1})$, and $A \in \mathcal{B}_{\tau^{n-1}}$, then

$$\mathbb{Q}_{\tilde{k}}(A \times X) = \mathbb{Q}_{\tilde{\ell}}(A). \tag{6.3}$$

By the Kolmogorov Extension Theorem, a consistent family of finite-dimensional distributions guarantees that a probability measure $\mathbb{P}$ on $\left(X^{\mathbb{N}}, \mathcal{B}[X^{\mathbb{N}}]\right)$ exists such that

$$\mathbb{Q}_{\tilde{k}}(A) = \mathbb{P}\left(\left\{x \in X^{\mathbb{N}} : (x_{k_1}, \ldots, x_{k_n})\right\}\right) \tag{6.4}$$

for all $A \in \mathcal{B}_{\tau^n}$ and $\tilde{k} = (k_1, \ldots, k_n)$ [50, 105, 112, 113].

The final step of the construction is to define $\mathbb{Q}_{\tilde{k}}$. Suppose $\tilde{k} = (k_1, \ldots, k_n)$ is ordered so that $k_i < k_j$ when $i < j$. Then define

$$\mathbb{Q}_{\tilde{k}}(A) = \int_{X^{k_1-1}} \int_{A_1} \cdots \int_{X^{k_n-k_{n-1}-1}} \int_{A_n} \prod_{i=1}^{k_n} \mathcal{G}\left[x_1^{i-1}, f\right](dx_i) \tag{6.5}$$

by integrating over the first k_n steps of the optimizer, restricted to the set A where required by the index set $\tilde{k}$. Here and below, the notation x_m^n refers to the trajectory in $\mathcal{T}[X]$ formed by concatenating $x_m, \ldots, x_n$, or to the empty trajectory if $n < m$. If $\tilde{k}$ is not ordered, define $\mathbb{Q}_{\tilde{k}}$ to meet the first consistency requirement above. That is, let $\tilde{\ell}$ be the ordered permutation of $\tilde{k}$ and set $\mathbb{Q}_{\tilde{k}} = \mathbb{Q}_{\tilde{\ell}}$. The family $\{\mathbb{Q}_{\tilde{k}}\}$ also satisfies the second requirement of consistency. If $\tilde{k} = (k_1, \ldots, k_n)$ with $n \geq 1$, $\tilde{\ell} = (k_1, \ldots, k_{n-1})$, and $A \in \mathcal{B}_{\tau^{n-1}}$, then the final integrals in Equation 6.5 are equal to one, and so

$$\begin{aligned} \mathbb{Q}_{\tilde{k}}(A \times X) &= \int_{X^{k_1-1}} \int_{A_1} \cdots \int_{X^{k_{n-1}-k_{n-2}-1}} \int_{A_{n-1}} \prod_{i=1}^{k_n} \mathcal{G}\left[x_1^{i-1}, f\right](dx_i) \\ &= \mathbb{Q}_{\tilde{\ell}}(A). \end{aligned} \tag{6.6}$$

Thus $\{\mathbb{Q}_{\tilde{t}}\}$ is a consistent family.

As a consequence, there exists a probability measure satisfying Equation 6.4. This measure governs the long-running outcome of the optimization process. For an optimizer $\mathcal{G}$ and an objective f, denote this measure by $\mathcal{G}_f$. The notation $\mathcal{G}f$ will also be used with equivalent meaning. Any random process Z distributed according to $\mathcal{G}_f$ is termed an *optimization process* of $\mathcal{G}$ on f, and $\mathcal{G}$ is said to *generate* Z on f.

Equation 6.5 gives the probability of an arbitrary cylinder set under $\mathcal{G}_f$. The marginal distribution of Z_n at any particular point in time can be stated more succinctly as $Z_n \sim \bigstar_{i=1}^{n} \mathcal{G}$. Conditional on $(Z_m)_{m=1}^{n-1}$, $Z_n \sim \mathcal{G}\left[(Z_m)_{m=1}^{n-1}, f\right]$. If g is a functional on $X^{\mathbb{N}}$, then $\mathbb{E}_{\mathcal{G}f}\left[g(Z)\right]$ is the expected value of the functional $g(Z)$ with respect to $\mathcal{G}_f$.

The space generated by infinitely extending one-step optimizers is considered next, followed by discussion of how these long-running optimizers may be integrated and whether they are continuous in objectives.

6.2 The Space of Long-Running Optimizers

In the previous section it was shown that for each optimizer $\mathcal{G} \in \mathcal{PF}$ and each objective f, there is a probability measure $\mathcal{G}_f$ that governs the infinite optimization sequence generated by $\mathcal{G}$. By examining Equation 6.5, it can be seen that the finite-dimensional distributions of a generalized optimizer $\mathcal{G} \in \mathcal{MF}$ are also well-defined and consistent. Thus generalized optimizers can also be infinitely expanded, but the signed measure $\mathcal{G}_f$ that results may not be finite.

For any $\mathcal{G} \in \mathcal{MF}$ that expands to a finite signed measure on any objective, consider the mapping $f \mapsto \mathcal{G}_f$. This mapping will be termed the *infinite expansion* of $\mathcal{G}$, and it will be represented simply as $\mathcal{G}_f$ or $\mathcal{G}f$. Thus the notation $\mathcal{G}_f$ can refer either to a measure over sequences for a specific f, or to the infinite expansion of $\mathcal{G}$ for a specific f. The infinite expansion contains all of the information necessary to run an optimizer $\mathcal{G}$ on an objective f.

Definition 6.2.1. *For any vector subspace $\mathcal{X} \subseteq \mathcal{MF}$, let*

$$\mathfrak{A}\left[\mathcal{X}\right] = \left\{f \mapsto \mathcal{G}_f : f \in X^{\mathbb{R}}, \mathcal{G} \in \mathcal{X}, \text{and}\, \forall h, \forall A,\ |\mathcal{G}_h(A)| < \infty\right\}$$

be the space of infinite expansions of $\mathcal{X}$ that result in a finite signed measure on $\left(X^{\mathbb{N}}, \mathcal{B}[X^{\mathbb{N}}]\right)$. If $\mathcal{G}_f \in \mathfrak{A}\left[\mathcal{X}\right]$, $\mathcal{G}_f$ is a function from $X^{\mathbb{N}}$ to $\mathcal{M}[X^{\mathbb{N}}]$.

The space of infinite expansions will be referred to as the space of *long-running optimizers* to distinguish it from $\mathcal{MF}$ and its subsets, which sample one point at a time. Long-running optimizers will be used in Chapter 9, where $\mathfrak{A}$ will be shown to be in duality with the space of function priors, extending and formalizing a result by Wolpert and Macready [218].

It is worthwhile to explore the relationship between $\mathcal{MF}$ and $\mathfrak{A}[\mathcal{MF}]$. Given any $\mathcal{G} \in \mathcal{MF}$ that expands to a finite measure, it is clear that the infinite expansion $\mathcal{G}_f \in \mathfrak{A}[\mathcal{MF}]$ is unique. The opposite does not hold true. Given $\mathcal{G}_f \in \mathfrak{A}[\mathcal{MF}]$, there is not a unique optimizer corresponding to it. There may even be uncountably many optimizers in $\mathcal{MF}$ that expand to $\mathcal{G}_f$. To see why, consider the following pseudo-optimizer:

$$\mathcal{G}[t, f](A) = \mathcal{G}_f \left(Z_{|t|+1} \in A \mid Z_1 = t^1, \ldots, Z_{|t|} = t^{|t|} \right). \tag{6.7}$$

If there were a unique $\mathcal{G}$ corresponding to $\mathcal{G}_f$, Equation 6.7 would define it. However, there is a major problem that prevents such a correspondence. There may be uncountably many sequences that have $\mathcal{G}_f$ measure zero, and thus there may also be uncountably many optimizers $\mathcal{G} \in \mathcal{MF}$ that extend to $\mathcal{G}_f$ that differ only on trajectories with $\mathcal{G}_f$-measure zero. In order to obtain a one-to-one correspondence between long-running optimizers and one-step-at-a-time optimizers, it is necessary to take the quotient space of $\mathcal{MF}$ consisting of equivalence sets on $\mathcal{MF}$ that are equal $\mathcal{G}_f$-almost surely, i.e. that are equal everywhere except on a set of trajectories that has $\mathcal{G}_f$-measure zero.

In general, a property holds $\mathcal{G}_f$-almost surely ($\mathcal{G}_f$-*a.s.*) if there is some subset A of $X^{\mathbb{N}}$ such that $\mathcal{G}_f(A) = 1$ and the property holds on A. In the following text, $\mathcal{G}_f$ will sometimes be treated as though it were a measure over trajectories in $\mathcal{T}[X]$. In this vein, a set of trajectories $T \subseteq \mathcal{T}[X]$ corresponds to the set of sequences in $X^{\mathbb{N}}$ that infinitely expand any trajectory in T. The set $T \subseteq \mathcal{T}[X]$ is described as having $\mathcal{G}_f$-measure zero if the set of all sequences that infinitely expand it has $\mathcal{G}_f$-measure zero. Also, if a property holds for all trajectories except on a set of $\mathcal{G}_f$-measure zero, then this property is said to hold $\mathcal{G}_f$-*a.s.*

The space of long-running optimizers is a vector space under pointwise addition and scalar multiplication. The vector structure of $\mathfrak{A}[\mathcal{MF}]$ is distinct from the vector structure of $\mathcal{MF}$ because addition and multiplication in $\mathfrak{A}$ are taken on measures over sequences, whereas addition and multiplication in $\mathcal{MF}$ are taken on measures over points. Thus if $\mathcal{G}_f = \alpha \mathcal{A}_f$, it does not follow that $\mathcal{G} = \alpha \mathcal{A}$. Nor does $\mathcal{G}_f = \mathcal{A}_f + \mathcal{B}_f$ imply that $\mathcal{G} = \mathcal{A} + \mathcal{B}$. In fact, such equalities hold only for trivial optimizers. There is also a norm for $\mathcal{A}$, given by $||\mathcal{G}_f||_{\mathfrak{A}} = \sup_{h \in X^{\mathbb{N}}} ||\mathcal{G}_h||_{\mathcal{M}}$. The vector subspace of long-running optimizers for which $||\mathcal{G}_f||_{\mathfrak{A}}$ is finite is a normed vector space. The infinite extensions of $\mathcal{PF}$ reside in this normed space.

6.3 Increasing Information and Stopping Times

This dissertation focuses on the space $\mathcal{PF}$, which consists of optimizers that sample from probability measures. The optimization process and its performance are therefore analyzed using the terminology and tools of stochastic processes. This section reviews background material necessary for understanding these concepts, particularly filtrations and stopping times. Filtrations capture the notion of increasing information, studied in Chapters 10 and 11. Some performance criteria in Chapter 7 will depend on integrating over stopping times. In particular, the No Free Lunch Theorems of Chapter 9 explicitly refer to the density of a stopped optimization process. These concepts will be defined in this section.

6.3.1 Filtrations and Information

One of the most important intuitions underlying the theory of stochastic processes is the concept of the σ-algebra as an information source. The σ-algebra has been referenced throughout this dissertation, yet up to this point these objects have been treated as arcane technical artifacts. In fact, the σ-algebra plays an intuitive role as a mediator of information. Let $\mathcal{F}$ be a σ-algebra. The distinct sets within the $\mathcal{F}$ represent observable events. Points that cannot be separated by sets in $\mathcal{F}$ are unobservable by any $\mathcal{F}$-measurable random variable. For example, suppose that x, z are distinct points in a measurable space $(X, \mathcal{F})$, and that every $A \in \mathcal{F}$ that contains x also contains z. Suppose Y is an arbitrary $\mathcal{F}$-measurable random variable. Then no observation of the state of Y can ever distinguish whether the state x or the state z has occurred; the σ-algebra $\mathcal{F}$ does not contain that information.

This dissertation has assumed that the search domain is a topological space (X, τ), and an optimizer applied to a particular objective and a particular evaluation history is a Borel measure, i.e. a measure on the Borel σ-algebra over τ, $\mathcal{B}_\tau$. Because the Borel σ-algebra contains all of the τ-open and τ-closed sets, it is the smallest set of information such that the boundary of every open and closed set is observable. That is, a Borel σ-algebra has sufficient information to determine when a trajectory within the search space has entered or exited a closed set.

The optimization process is $\mathcal{B}[X^{\mathbb{N}}]$-measurable. As the optimization process unfolds, it generates information about which sequences in $X^{\mathbb{N}}$ are possible. Each step of the optimizer restricts one component of the sequence. The future steps of the optimizer can never be distinguishable by the optimization process, even though these future steps are $\mathcal{B}[X^{\mathbb{N}}]$-measurable. Every prefix of the optimization process is therefore measurable with respect to a smaller σ-algebra that contains only the information generated up to the present. The sequence of such σ-algebras for any process is termed the *natural filtration* of the process.

A filtration represents a sequence of increasing information. As more information is acquired, more events become observable. Formally, a filtration on a measurable space $(X, \mathcal{F})$ is a sequence of σ-algebras $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ such that $\mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F}$ for all n. A stochastic process $Y = (Y_n)_{n\in\mathbb{N}}$ is *adapted* to the filtration $\{\mathcal{F}_n\}$ if Y_n is an $\mathcal{F}_n$-measurable random variable for all n. The natural filtration of a process is the smallest filtration to which it is adapted, and it encapsulates the information that the process has acquired at each point in time. Filtrations on the search domain will play a key role in Chapter 11, where a filtration will be constructed in order to direct the optimization process towards the global optimum.

Denote by $\mathcal{Z}_n$ the natural filtration of the optimization process on $\left(X^{\mathbb{N}}, \mathcal{B}[X^{\mathbb{N}}]\right)$. Then each Z_n is $\mathcal{Z}_n$-measurable. For any objective function f, the stochastic process $E_n = f(Z_n) - f^*$ is the error process of the optimizer on f, recalling that f^* is the minimum of f. The error process is adapted to $\mathcal{Z}_n$. It will be explored in more detail in the next chapter as a means of assessing the performance of an optimizer.

6.3.2 Stopping Times

A *random time* of a discrete stochastic process on $(X, \mathcal{F})$ is a random index of the process that is $\mathcal{F}$-measurable. That is, if T is a random time, then the sets $\{T \leq n\}$ and $\{T > n\}$ are elements of $\mathcal{F}$ for all n. Given the entire process, the value of the random time can be determined exactly. Given only a prefix of the process, it is not necessarily possible to know whether or not the random time has passed at the current time.

A *stopping time* of a process Y with respect to a filtration $\{\mathcal{F}_n\}$ is a random time that is measurable with respect to the filtration for each value it takes on. That is, if S is a stopping time, $\{S \leq n\} \in \mathcal{F}_n$ for all n. With a stopping time, it is possible to determine when the stopping time has occurred based on the value of the process up to the current time.

In the next chapter, stopping times will be used as criteria for deciding when to measure the performance of an optimizer. They will also show up again in Chapter 9 in the proof of the No Free Lunch Identification Theorem (Theorem 9.3.7), which depends on an integral of the optimization process up to a stopping time.

A stopping time need not be finite. In that case, $\{S = \infty\}$ is an element of $\mathcal{F}$, but does not usually appear in any of the $\mathcal{F}_n$. If the stopping time is finite on a particular sequence in the state space, then it is said to *hit* on that sequence. One cannot always determine whether a stopping time will hit on a particular sequence given only a finite number of components. The possibility that $S = \infty$ must always be considered.

For any stopping time S of a process Y, there is a random variable Y_S representing the value of the process at the stopping time. The variable Y_S is measurable with respect to the σ-algebra $\mathcal{F}_S \equiv \{A \in \mathcal{F} : A \cap \{S \leq n\} \in \mathcal{F}_n \, \forall n\}$.

Suppose that S is a stopping time of the optimization process Z. S is a function over sequences, i.e. $S = S(x)$ for $x \in X^{\mathbb{N}}$; it will also be written as a function over trajectories, $S = S(t)$, when it is possible to do so, i.e., when $S(t) \leq |t|$. The quantity Z_S is a random variable on $\mathcal{Z}_S$ representing the evaluation point on which S hits. We need to know how to integrate over the joint probability of the sequence $Z_1, \ldots, Z_S$. Let H_n be the set of trajectories on which S has hit by time n, i.e. $H_n = \{t \in \mathcal{T}[X] : |t| = S(t) = n\}$. Let $H = \bigcup_n H_n$, the set of finite stopping trajectories for S. If $S < \infty$ $\mathcal{G}_f$-almost surely, i.e. $\mathcal{G}_f(\{S = \infty\}) = 0$, then the joint distribution of $Z_1, \ldots, Z_S$ is given by

$$\mathcal{G}_f(A) = \sum_{n=1}^{\infty} \int_{H_n \cap A} \prod_{i=1}^{n} \mathcal{G}[t_1^{i-1}, f](dt^i) \tag{6.8}$$

for any $A \in \mathcal{Z}_S$. For any functional h_S that is finitely determined by S, i.e. $h_S(Z) = h_S(Z_1, \ldots, Z_S)$, the expected value of h_S is given by

$$\mathbb{E}_{\mathcal{G}f}[h_S(Z)] = \sum_{n=1}^{\infty} \int_{H_n} h_S(t) \prod_{i=1}^{n} \mathcal{G}[t_1^{i-1}, f](dt^i). \tag{6.9}$$

In this equation, a stopping time has been used to reduce an integral over infinite sequences in $X^{\mathbb{N}}$ to an integral over finite trajectories in $\mathcal{T}[X]$. This reduction is useful because there are significantly more mathematical tools available to deal with infinite sums than with infinite products. Also notice that once the functional is finitely determined as in Equation 6.9, the results of Theorems 6.4.2 and 6.4.4 below can be applied to demonstrate the continuity of $\mathbb{E}_{\mathcal{G}f}[h_S(Z)]$ in objectives and optimizers.

6.4 Continuity of the Optimization Process

In order to analyze optimizer performance, it will be important to answer the following question: When does continuity of an optimizer imply that the optimization process generated by that optimizer is continuous? Specifically, suppose $f_n \to f$, and let $\mathcal{G} \in \mathcal{PF}$ be continuous in objectives. Does $\mathcal{G}_{f_n} \to \mathcal{G}_f$ in the norm topology of $\mathfrak{A}[\mathcal{MF}]$? Because the optimization process is infinite, it may be possible for $\mathcal{G}_{f_n}$ to diverge from $\mathcal{G}_f$ even if $\mathcal{G}$ is continuous everywhere. Thus it is not possible to extend continuity in $\mathcal{MF}$ to the norm topology of $\mathfrak{A}$.

It is possible to prove that the continuity of an optimizer on sufficiently many trajectories implies that the long-running optimizer results in similar average values for *finitely-determined* random variables of the optimization process:

Definition 6.4.1. *A random variable $Y(Z)$ defined over the optimization process is finitely determined if there exists a fixed $m < \infty$ such that $Y(Z) = \tilde{Y}(Z_1, \ldots, Z_m)$.*

It will be shown that for any optimizer $\mathcal{G} \in \mathcal{MF}$ that is continuous $\mathcal{G}_f$-*a.s.*,

$$\mathbb{E}_{\mathcal{G}f_n}[Y(Z)] \to \mathbb{E}_{\mathcal{G}f}[Y(Z)], \tag{6.10}$$

for any finitely-determined random variable Y. The condition of finiteness is needed because the infinitesimal differences between $\mathcal{G}_{f_n}$ and $\mathcal{G}_f$ can cause divergence of the integral after infinitely many time steps.

Notice the use of the expectation operator $\mathbb{E}$ even though the optimizer $\mathcal{G}$ was stated to be in the space $\mathcal{MF}$. In this case, the operator $\mathbb{E}$ is used to signify an integral over the whole space $X^{\mathbb{N}}$, and $\mathcal{G}_f$ need not be a probability measure. The symbol $\mathbb{E}$ is used nonetheless because the focus of the text is on probability measures.

The space of random variables on $\left(X^{\mathbb{N}}, \mathcal{B}[X^{\mathbb{N}}]\right)$ is the set of functionals on $h \in X^{\mathbb{N}} \to \mathbb{R}$ whose backward projections are $\mathcal{B}[X^{\mathbb{N}}]$-measurable, that is, $h^{-1}(A) \in \mathcal{B}[X^{\mathbb{N}}]$ for every A in the Borel σ-algebra on $\mathbb{R}$. These random variables will be written either in lower case as $h(Z)$ or in upper case as $H(Z)$. If written in upper case, the argument may be omitted, e.g. $H = H(Z)$.

If $g(Z)$ is a random variable of this sort, then $\mathbb{E}_{\mathcal{G}f}\left[g(Z)\right]$ integrates over X countably many times. But if g is finitely determined, it depends on only finitely many components in $X^{\mathbb{N}}$. The remaining (infinitely many) steps can be integrated out. Such a variable is said to be *finitely determined*. If x_1^m is the trajectory formed by taking the first m components of $x \in X^{\mathbb{N}}$ and $g(x) = g(x_1, \ldots, x_m)$, m integrals are required, since

$$\begin{aligned}
\mathbb{E}_{\mathcal{G}f}\left[g\left(Z_1, \ldots, Z_m\right)\right] &= \int_{X^{\mathbb{N}}} g(x_1, \ldots, x_m) \prod_{k=1}^{\infty} \mathcal{G}\left[x_1^{k-1}, f\right](dx_k) && (6.11)\\
&= \int_{X^m} g(x_1, \ldots, x_m) \prod_{k=1}^{m} \mathcal{G}\left[x_1^{k-1}, f\right](dx_k)\\
&\quad \times \prod_{j=m+1}^{\infty} \int_X \mathcal{G}\left[x_1^{j-1}, f\right](dx_j) && (6.12)\\
&= \int_{X^m} g(x_1, \ldots, x_m) \prod_{k=1}^{m} \mathcal{G}\left[x_1^{k-1}, f\right](dx_k). && (6.13)
\end{aligned}$$

Along any particular trajectory t, the optimization processes of $\mathcal{G}_f$ and $\mathcal{G}_{f_n}$ cannot move far apart when $\mathcal{G}$ is continuous in objectives on the trajectory t. If $\mathbb{E}_{\mathcal{G}f}\left[h(Z)\right]$ depends on finitely many optimization steps, then for large n, $\mathbb{E}_{\mathcal{G}f}\left[h(Z)\right]$ must be close to $\mathbb{E}_{\mathcal{G}f_n}\left[h(Z)\right]$ as well if $\mathcal{G}$ is continuous in objectives at f. In fact, $\mathcal{G}$ need not be continuous at every trajectory; it is enough if $\mathcal{G}$ is continuous at f on a large enough set of trajectories. In this case, "large enough" means that $\mathcal{G}$ must be continuous in objectives at f for a set of trajectories that has full measure on $\mathcal{G}_f$. That is, $\mathcal{G}$ must be *continuous* $\mathcal{G}_f$*-a.s.*

For the evolutionary and quasi-evolutionary algorithms of Chapter 5, trajectories of ambivalent fitness (Definition 5.3.4) must have zero probability of occurring when $\mathcal{G}$ is run. The only trajectories on which many evolutionary and quasi-evolutionary algorithms are discontinuous are the trajectories of ambivalent fitness. If $\mathcal{G}$ is a population-based algorithm with population size K, the following theorem gives the condition under which $\mathcal{G}$ will be continuous $\mathcal{G}_f$-*a.s.* on f. Basically, the optimizer must place probability zero on points that would extend a trajectory ambivalently.

Theorem 6.4.1. *Let f be an objective, and let $\mathcal{G} \in \mathcal{PBO}_K$ be an optimizer that is continuous in objectives on t, f for all trajectories t of unambivalent fitness on f. Let A_t be the set of points in X for which $t \cup x$ is of ambivalent fitness on f. Then $\mathcal{G}$ is continuous $\mathcal{G}_f$-a.s. on f if $\mathcal{G}[t, f](A_t) = 0$ for all trajectories t of unambivalent fitness on f.*

Proof. The proof is by induction on the length of the trajectory. Every trajectory of length 1 is of unambivalent fitness. Suppose that trajectories of length $n-1$ are unambivalent with probability one. Let t be an arbitrary trajectory of length $n-1$. Let $U = X \setminus A_t$ so that $\mathcal{G}[t, f](U) = 1$. That is, extensions of t to length n are unambivalent with probability one. Since t was arbitrary, trajectories of length n are in general unambivalent with probability one. Therefore trajectories of arbitrary length are of unambivalent fitness on f with probability one, i.e., $\mathcal{G}$ is $\mathcal{G}_f$-*a.s* continuous. $\square$

This theorem is sufficient to prove the $\mathcal{G}_f$-*a.s.* continuity of evolutionary algorithms in many cases. For example, if the search domain is d-dimensional

Euclidean space, $X = \mathbb{R}^d$, then a real-coded genetic algorithm with tournament selection, masked crossover, and Gaussian selection is $\mathcal{G}_f$-*a.s.* continuous on objective functions without fitness plateaus, that is, on all objective functions whose level sets have Lebesgue measure zero.

The performance of an optimizer is a function of the optimizer and the objective on which it is run. Roughly, the overall performance of an optimizer is the weighted average of its performance on every possible run of the optimizer. This average can be found be integrating over $\mathcal{G}_f$. It is important to know whether average performance changes only slightly when the optimizer or the objective function are altered slightly. The next theorem shows that if an optimizer is continuous $\mathcal{G}_f$-*a.s.* in objectives, then the expected value of finitely determined random variables changes continuously with the objective. If the performance of an optimizer is assessed after finitely many optimization steps, this next theorem will imply that the average performance should not change much if the objective is not changed much.

Theorem 6.4.2. *Let $\mathcal{G} \in \mathcal{MF}$ be continuous $\mathcal{G}_f$-a.s. at an objective f, and let $f_n \to f$ pointwise. Let $g\left(x_1, \dots, x_m\right)$ be a real function on X^m with $m < \infty$ fixed, and suppose that $\mathbb{E}_{\mathcal{G}f}\left|g\left(Z_1, \dots, Z_m\right)\right| < \infty$ and $\mathbb{E}_{\mathcal{G}f_n}\left|g\left(Z_1, \dots, Z_m\right)\right| < \infty$. Then $\mathbb{E}_{\mathcal{G}f_n}\left[g\left(Z_1, \dots, Z_m\right)\right] \to \mathbb{E}_{\mathcal{G}f}\left[g\left(Z_1, \dots, Z_m\right)\right]$.*

Proof. Fix $\epsilon > 0$. Assume $||\mathcal{G}[t, f]||_{\mathcal{M}} \leq M < \infty$. Suppose J and L are two index sets of positive integers less than or equal to m. J and L will be termed *complementary* if $J \cap L = \emptyset$ and $J \cup L = \{1, \dots, m\}$. Let $\mathcal{K}$ be the set of all complementary pairs of index sets. There are exactly 2^m such pairs. These complementary sets can be used to state the joint distribution of $Z_1, \dots, Z_m$ as a sum.

Let t be an unambivalent trajectory of length at least $m < \infty$. Recall that t_1^m is the trajectory formed by taking the first m components of t. Then

$$\begin{aligned}\prod_{k=1}^{m} \mathcal{G}\left[t_1^{k-1}, f\right]\left(dt^k\right) &= \prod_{k=1}^{m}\left[\left(\mathcal{G}\left[t_1^{k-1}, f\right]\left(dt^k\right) - \mathcal{G}\left[t_1^{k-1}, f_n\right]\left(dt^k\right)\right)\right. \\ &\qquad \left. + \, \mathcal{G}\left[t_1^{k-1}, f_n\right]\left(dt^k\right)\right] \qquad (6.14) \\ &= \sum_{J,L\in\mathcal{K}}\left[\prod_{j\in J}\left(\mathcal{G}\left[t_1^{j-1}, f\right]\left(dt^j\right) - \mathcal{G}\left[t_1^{j-1}, f_n\right]\left(dt^j\right)\right)\right. \\ &\qquad \left. \times \prod_{\ell\in L}\mathcal{G}\left[t_1^{\ell-1}, f_n\right]\left(dt^\ell\right)\right]. \qquad (6.15)\end{aligned}$$

Equation 6.15 expands the product in Equation 6.14 by cross multiplying the difference with the joint distribution over f_n. This sum contains 2^m terms, one for each pair of complementary index sets. With the exception of the complementary sets given by $J_0 = \emptyset, L_0 = \{1, \ldots, m\}$, every pair of complementary index sets in $\mathcal{K}$ yields a product in Equations 6.15 with at least one factor of the form

$$\mathcal{G}\left[t_1^{j-1}, f\right]\left(dt^j\right) - \mathcal{G}\left[t_1^{j-1}, f_n\right]\left(dt^j\right).$$

Because m is finite and t is fixed and of unambivalent fitness, it is possible to choose n so that $\left|\mathcal{G}[t_1^{j-1}, f] - \mathcal{G}[t_1^{j-1}, f_n]\right| < \frac{\epsilon}{2^{2m}M^m}$ for each j. Thus each term in the sum except for the one at J_0, L_0 is less than $\frac{\epsilon}{2^m}$, since $\mathcal{G}$ is of bounded magnitude M. Further, the term in the sum at J_0, L_0 reduces to

$$\prod_{k=1}^{m} \mathcal{G}\left[t_1^{k-1}, f_n\right]\left(dt^k\right),$$

and therefore for $A \in \mathcal{B}_{\tau^m}$,

$$\begin{aligned}&\int_A \left|\prod_{k=1}^{m} \mathcal{G}\left[t_1^{k-1}, f\right]\left(dt^k\right) - \prod_{k=1}^{m} \mathcal{G}\left[t_1^{k-1}, f_n\right]\left(dt^k\right)\right| \\ &\qquad \leq \sum_{J,L\in\mathcal{K}\setminus\{J_0,L_0\}} \int_A \prod_{j\in J}\left|\mathcal{G}\left[t_1^{j-1}, f\right]\left(dt^j\right) - \mathcal{G}\left[t_1^{j-1}, f_n\right]\left(dt^j\right)\right| \\ &\qquad < 2^m \frac{\epsilon}{2^{2m}M^m} 2^m M^m = \epsilon. \qquad (6.16)\end{aligned}$$

Because of the integrability assumptions on g, it follows that

$$\left|\mathbb{E}_{\mathcal{G}f_n}\left[g\left(Z_1,\ldots,Z_m\right)\right]-\mathbb{E}_{\mathcal{G}f}\left[g\left(Z_1,\ldots,Z_m\right)\right]\right|\to 0. \tag{6.17}$$

$\square$

Corollary 6.4.3. *Under the same general assumptions as Theorem 6.4.2, let A be a set in $\mathcal{B}[X^{\mathbb{N}}]$ such that for fixed $m<\infty$, A is independent of Z_n for $n>m$ under $\mathcal{G}_f$ and $\mathcal{G}_{f_n}$. Then $\mathcal{G}_{f_n}(A)\to\mathcal{G}_f(A)$.*

Proof. Note that $\mathcal{G}_f(A)=\mathbb{E}_{\mathcal{G}f}[1_A]$. Define $g(Z_1,\ldots,Z_m)=\mathbb{E}_{\mathcal{G}f}[1_A \mid Z_1,\ldots,Z_m]$. Because A is independent of Z_n for $n>m$, $g(Z_1,\ldots,Z_m)=1_A(Z)$ by the definition of conditional expectations. The result follows directly from Theorem 6.4.2. $\square$

If the objective is held constant, but the optimizer is altered slightly, a similar theorem holds without continuity assumptions. Integrals over finitely determined random variables change continuously with the optimizer, regardless of whether the optimizer is continuous. The next theorem shows that the average value of a functional under $\mathcal{G}_n f$ converges to its average value under $\mathcal{G}f$, again if the functional depends on finitely many steps of the optimization process. This result will be used to demonstrate that performance criteria are continuous over optimizers.

Theorem 6.4.4. *Let $\mathcal{G}\in\mathcal{MF}$, and let $f\in X^{\mathbb{N}}$. Let $\mathcal{G}_n\to\mathcal{G}$ under the norm $||\cdot||_{\mathcal{MF}}$. Let $g\left(x_1,\ldots,x_m\right)$ be a real function with $m<\infty$ fixed, and suppose that $\mathbb{E}_{\mathcal{G}f}\left|g\left(Z_1,\ldots,Z_m\right)\right|<\infty$ and $\mathbb{E}_{\mathcal{G}_nf}\left|g\left(Z_1,\ldots,Z_m\right)\right|<\infty$. Then $\mathbb{E}_{\mathcal{G}_nf}\left[g\left(Z_1,\ldots,Z_m\right)\right]\to\mathbb{E}_{\mathcal{G}f}\left[g\left(Z_1,\ldots,Z_m\right)\right]$*

Proof. Repeat the proof of Theorem 6.4.2, replacing $\mathcal{G}[t_1^{k-1},f_n]$ by $\mathcal{G}_n[t_1^{k-1},f]$. $\square$

Theorem 6.4.2 and 6.4.4 are sufficient to prove the continuity of performance criteria on continuous optimizers, which is done in Chapter 7.

6.5 Conclusion

This chapter discussed the properties of the infinite optimization process, which is well-defined for every optimizer in $\mathcal{PF}$. It also briefly introduced the relevant background in stochastic processes that will be used to establish the theorems of subsequent chapters. Finally, it has been shown that long-running optimizers weakly preserve continuity in the sense that the expected value of finitely-determined random variables converges when applied to similar objectives and optimizers. The next chapter develops the analysis of performance based on the results of this chapter.

Chapter 7

Performance Analysis

Given a particular objective function to be optimized, it would be useful to know which optimizer will perform best on that objective. Indeed, the entire purpose of studying the space of optimizers is to provide tools to answer this very question. To this end, different categories of performance criteria are analyzed theoretically in this chapter. Many performance criteria are shown to be continuous and non-linear, implying that similar optimizers perform similarly and that linearly interpolated optimizers may outperform the optimizers being interpolated. These facts are demonstrated experimentally in Chapter 8. Further, the categories of performance criteria described in this chapter make it possible to identify the conditions under which No Free Lunch theorems hold in infinite-dimensional spaces, to be undertaken in Chapter 9.

7.1 Performance Criteria

This section introduces *performance criteria* that formalize common notions of what it means for an optimizer to perform well on an objective. A performance criterion takes an optimizer and an objective function and outputs a real number, providing an objectively determined score for each optimizer on each cost function. As a convention, this score is required to be nonnegative, and a value of zero is considered perfect performance.

Most of the performance criteria considered here are defined with respect to the error magnitude at each optimizer step.

Definition 7.1.1. *The error sequence* $E(z) = (E_n(z))_{n\in\mathbb{N}}$ *of a sequence* $z \in X^{\mathbb{N}}$ *on an objective* f *is the sequence on* $\mathbb{R}^{\mathbb{N}}$ *given by*

$$E_n(z) = f(z_n) - f^* \tag{7.1}$$

for any f that is bounded below, i.e. $f^* > -\infty$. When the objective function must be stated explicitly, the error sequence may be written as $E^f(z)$ or $E_n^f(z)$.

The error sequence of the optimization process, $E(Z)$, will be termed the *error process*. It is adapted to the natural filtration of the optimization process, $\{\mathcal{Z}_n\}$. The sequence of evaluation points along the optimization process that corresponds to the sequence of best evaluation points so far will be termed the *running minimum process*, denoted by Z^*, with $Z_n^* = \text{argmin}_{\{Z_m : m \leq n\}} f\left(Z_m\right)$. That is, Z_n^* is the best known solution at time n. Define the *minimum error sequence* $E^*(z)$ as the running minimum of the error sequence,

$$E_n^*(z) = \min_{m \leq n} E_n(z) = f(Z_n^*) - f^*. \tag{7.2}$$

The *minimum error process* is the minimum error sequence of the optimization process, $E^*(Z)$. These definitions will be used to define classes of performance criteria.

A *performance criterion* is defined as the expected value of a positive functional of the optimization process.

Definition 7.1.2. *Let $\mathcal{G} \in \mathcal{PF}$ and $f \in X^{\mathbb{R}}$, and let $Z = \left(Z_n\right)_{n \in \mathbb{N}}$ be the optimization process of $\mathcal{G}$ on f. Then a function $\phi : \mathcal{PF} \times \mathbb{R}^X \to [0, \infty)$ is a performance criterion if there exists a function $h : X^{\mathbb{N}} \times \mathbb{R}^X \to [0, \infty)$ with appropriate measurability properties such that*

$$\phi\left(\mathcal{G}, f\right) = \mathbb{E}_{\mathcal{G}f}\left[h(Z, f)\right] = \int_{X^{\mathbb{N}}} h(z, f)\, \mathcal{G}_f(dz) \tag{7.3}$$

whenever the integrals exist. More generally, ϕ may be extended to $\mathcal{MF}$ using the integral on the right.

Performance criteria can be used to compare optimizers to each other, and to analyze how the performance of an optimizer varies as the objective changes. Ultimately, an analysis of performance should reveal how to select a particular optimizer for a particular task. This issue will be approached experimentally in the next chapter and theoretically in Chapters 9 and 10.

The remainder of this section gives examples of possible performance criteria that correspond broadly to the kinds of results reported in the experimental literature on optimizers. These examples are given in four groups: (1) evaluation by average error, (2) hitting times for an error bound, (3) probability of attaining an error bound, and (4) error at a stopping time.

7.1.1 Evaluation by Average Error

A first approach to evaluating optimizers is to average the magnitude of the errors the optimizer makes at each time step. This metric combines the total accuracy along with the speed of convergence, at the risk of disproportionately penalizing optimizers for early errors due to exploration of the objective. Such a metric is not traditionally reported, but could prove useful, since it contains information about the convergence speed of the optimizer.

Let $f \in \mathbb{R}^X$, $\mathcal{G} \in \mathcal{PF}$, and let $Z = (Z_n)$ be the optimization process generated by $\mathcal{G}$ on f. Define a performance criterion by

$$\phi_w \left(\mathcal{G}, f\right) = \mathbb{E}_{\mathcal{G}f} \left[\sum_{n=1}^{\infty} w_n \left| f\left(Z_n^*\right) - f^* \right| \right], \tag{7.4}$$

where w_n is a sequence of weights that can be used to discount later values. Three basic choices for w_n are (1) $w_n = 1$, which treats all errors equally but only results in ϕ_w finite when $\mathcal{G}$ converges on f at a fast enough rate, (2) $w_n = 2^{-n}$, which places more weight on earlier errors but is finite whenever the objective is almost surely finite on $\mathcal{G}[\emptyset, f]$, and (3) $w_n = 1$ for $n \leq N$ for some fixed $N < \infty$ and zero otherwise, which considers only a finite number of time steps. Another possible scheme might ignore initial errors up to a finite time, allowing optimizers to explore more broadly in earlier stages without penalty.

The function ϕ_w using any of the three methods described above has two primary advantages. First of all, it captures a natural intuition for evaluating an optimizer, namely, the magnitude of errors it makes before finding a good optimum. Secondly, by taking a sum of these errors, ϕ_w measures the convergence rate of an optimizer. The disadvantage of ϕ_w is that it can be sensitive to the early errors of an optimizer, especially when $w_n = 2^{-n}$. Also, if w_n is set according to either the second or third option above, then later errors will be ignored, and an asymptotically convergent optimizer that converges late will be outscored by a non-convergent optimizer that attains good but suboptimal solutions earlier on (which may or may not be a desirable feature).

One may wish to estimate the value of a performance criterion in order to evaluate various optimizers. If w_n is set according to the first option ($w_n = 1$ for all n), then there is no reliable way to approximate the value of ϕ_w through sample runs. No matter how many times an optimizer converges to the correct solution, it is always possible that there is a set of sample runs with positive probability on which the algorithm never reaches the global optimum. In this case, the integrand is infinite on a set of positive probability, and thus it is possible to have $\phi_w = \infty$ even if the cumulative error appears small and finite for all observed runs. In fact, many optimizers of interest will have $\phi_w = \infty$ on a large number of problems (e.g. most genetic algorithms). Thus setting $w_n = 1$ for all n is practically undesirable unless one has a proof that an algorithm converges in probability to the global optimum on all objectives of interest.

If w_n is set according to either $w_n = 2^{-n}$ or $w_n = 1$ for $n \leq N$, then the value of $\phi_w(\mathcal{G}, f)$ can be estimated using Monte Carlo methods by running several instances of the optimizer $\mathcal{G}$ on f for a fixed number of iterations. In the first case, the number of iterations is chosen to satisfy a tolerance, $2^{-N} < \epsilon$; in the second, the number of iterations is simply the bound N. The minimum error sequence $E^*(Z)$ is non-increasing, and thus ϕ_w converges for either choice of w provided that $E_1^*(Z)$ is finite with probability one for all n.

In Chapter 8, results will be reported for both $w_n = 2^{-n}$ and $w_n = 1$ for $n < N$.

7.1.2 Evaluation by Hitting Time

In existing literature, when evaluating a proposed optimizer, the optimizer is often run on a benchmark set of problems for which the optima are known (see e.g. [10, 34, 80]). A common performance criterion for ranking optimizers is to count the number of points that must be generated before obtaining a solution whose fitness is within a fixed error from the globally optimal fitness.

For a fixed error $\epsilon > 0$, define the hitting time for ϵ as the first time when an evaluation point has global error less than ϵ, i.e. $\tau_\epsilon \equiv \min\{n : |f(Z_n) - f^*| \leq \epsilon\}$. Then define a performance criterion by

$$\psi_\epsilon(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}[\tau_\epsilon], \tag{7.5}$$

which is the average hitting time for ϵ over all runs of the algorithm $\mathcal{G}$ on the objective f.

This formula has a serious flaw for non-convergent optimizers. If $\mathcal{G}$ has a positive probability of failing to attain error less than ϵ, then $\psi_\epsilon = \infty$. Additionally, from the standpoint of approximation, only finite computational time is available, and thus cases in which τ_ϵ is large cannot be distinguished computationally from cases in which it is infinite.

One alternative is to place a finite limit on the stopping time; that is, for $N < \infty$,

$$\psi_\epsilon^N(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}[\tau_\epsilon \wedge N], \tag{7.6}$$

where the notation $\tau_\epsilon \wedge N = \min\{\tau_\epsilon, N\}$ as usual. The criterion $\psi_\epsilon^N(\mathcal{G}, f)$ can be estimated reasonably by running $\mathcal{G}$ on f several times for at most N evaluations. This performance criterion also reflects a natural criterion for comparing optimizers; it measures the average number of steps the optimizer must be run before it produces a solution correct within error ϵ. Unlike ϕ_w, ψ_ϵ^N is generally bounded across optimizers and objectives; optimizers will have $\psi_\epsilon^N \leq N$ on all objectives. Unfortunately, ψ_ϵ and ψ_ϵ^N are discontinuous over objective functions, as will be discussed below.

7.1.3 Evaluation by Success Probability

The hitting time tests how long it takes on average to attain an error threshold ϵ. However, it does not test how often the threshold is attained. Define the sets $T_\epsilon = \{\tau_\epsilon < \infty\}$ and $T_\epsilon^N = \{\tau_\epsilon < N\}$ to represent respectively the sequences that asymptotically attain a given error bound and those that attain it within a fixed number of evaluations. Then the *success probability* is the probability of attaining a bound asymptotically, and the *finite success probability* is the probability of attaining the bound within a finite time window [210]. Each of these are performance criteria given by

$$\sigma_\epsilon(\mathcal{G}, f) = \mathcal{G}_f(T_\epsilon), \quad \sigma_\epsilon^N(\mathcal{G}, f) = \mathcal{G}_f(T_\epsilon^N). \tag{7.7}$$

To see that σ_ϵ and σ_ϵ^N are performance criteria, recall that $\mathcal{G}_f(A) = \mathbb{E}_{\mathcal{G}f}[1_A(Z)]$ where 1_A is the indicator set of A, i.e. $1_A(x) = 1$ if $x \in A$ and is zero otherwise. The finite success probability is the preferred criterion, since σ_ϵ^N can be estimated experimentally, whereas σ_ϵ cannot. Notice that σ_ϵ does not conform to the convention that lower performance values should be better and zero should be optimal. The convention is ignored here because the success probability has an intuitive meaning in its own right. In situations where the convention is important, the performance criterion $1 - \sigma_\epsilon$ can be used instead.

Given the finite success probability, it is of interest to know the average hitting time for sequences that attain the error bound. The average hitting time on successful trajectories is a performance criterion, given by

$$\hat{\psi}_\epsilon^N(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[(\tau_\epsilon \wedge N)\, 1_{T_\epsilon}(Z)\right]. \tag{7.8}$$

On its own, this quantity is not useful, since it may be zero when the optimizer fails, i.e. when $\mathcal{G}_f(T_\epsilon^N) = 0$. However, the pair $\left(\hat{\psi}_\epsilon^N, \sigma_\epsilon^N\right)$ disambiguates this situation, and these two values can be reported together for completeness [10].

7.1.4 Evaluation by Error at a Stopping Time

Optimizers are often tested by running the algorithm for a fixed number of evaluations and then reporting the final error. As a generalization of this type of evaluation, suppose that an optimizer is run until some criterion is satisfied, not necessarily connected to the number of evaluations. As one example of why this generalization may be useful, suppose that rather than stopping after a fixed number of evaluations, one wishes to stop an optimizer after it uses up a fixed amount of resources, such as CPU cycles or calendar time. Such a criterion can be modeled as a stopping time, and the error magnitude at this stopping time is a performance criterion.

Let T be a stopping time equal to the generation in which this resource limit is first expended, and define a performance criterion by

$$\zeta_T(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f} \left| f\left(Z_T^*\right) - f^* \right|, \tag{7.9}$$

so that ζ_T is the smallest error attained within the allocated resources, where Z_n^* is the running minimum on Z_n as above.

One stopping time that will be used extensively is the number of unique points evaluated. In an environment where function evaluation is expensive, the objective value of repeated points can be retrieved from a cache. In this case, it is reasonable to suggest that repeated evaluation points are irrelevant to overall performance. Given a sequence $z \in X^{\mathbb{N}}$, let

$$T_m(z) = \min\left\{n \in \mathbb{N} \mid z_1, \ldots, z_n \text{ contains } m \text{ unique points}\right\}. \tag{7.10}$$

Performance criteria based on T_m are used to derive No Free Lunch theorems in Chapter 9, extending previous results of this type that only applied to optimizers that never repeat any point.

Performance criteria defined on this sequence will be studied almost exclusively from this point, and thus it is worthwhile to define this sequence independently.

Definition 7.1.3. *The sequence of stopping times given by $(T_m)_{m=1}^{\infty}$ is termed the unique stopping sequence.*

As a variation on ϕ_w above, one may define the average minimum error after each unique individual by

$$\phi_T(\mathcal{G}, f) = \sum_{m=L}^{U} \zeta_{T_m}(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[\sum_{m=L}^{U} \left| f\left(Z_{T_m}^*\right) - f^* \right|\right] \tag{7.11}$$

for some lower bound $L \geq 1$ and upper bound $U < \infty$. The criterion ϕ_T is finite whenever $E_{T_L}^* = f\left(Z_{T_L}^*\right) - f^*$ is finite with probability one.

Usually, it is not difficult to estimate ζ_{T_m} or ϕ_T. Most optimizers produce unique points with some frequency, so that $T_m < \infty$ almost surely if $m < |X|$. In infinite spaces, it is even common to have $T_m = m$ $\mathcal{G}_f$-*a.s.* If an optimizer does not produce m unique points, or does so slowly, this property of the optimizer will generally be known ahead of time either analytically or constructively. If $T_m = \infty$, then the set of unique points in the optimization process is of size at most $m-1$, so $E^*_{T_m} = E^*_{T_{m-1}}$ and $\zeta_{T_m} = \zeta_{T_{m-1}}$. Thus the infinite case is easy to handle when it can be identified. It is only difficult to approximate ζ_{T_m} when unique points are generated slowly. In this case, assuming that $T_m = \infty$ will produce an overestimate of the performance criterion. Optimizers that produce unique points slowly are generally undesirable, and thus an overestimate of the performance criterion for these optimizers is not problematic.

A substantial number of performance criteria have now been introduced. The next two sections discuss the mathematical properties of performance criteria, such as nonlinearity, decomposability, and continuity.

7.2 Properties of Performance Criteria

It is clear that a wide variety of performance criteria exists. These criteria can be analyzed in general according to their mathematical properties. This section examines three such properties that a performance criterion may possess: (1) nonlinearity, (2) progressive decomposability, and (3) dependence on the error sequence. The question of continuity in performance criteria is a larger topic and will be addressed separately in the next section.

7.2.1 Nonlinearity

All non-trivial performance criteria are nonlinear in both arguments. A performance criterion is trivial if it does not depend on the optimizer, i.e. $\phi(\mathcal{G}, f) = \phi(f)$, or if it only depends on the first element of the error sequence, i.e. $\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[g\left(E_1\left(Z\right)\right)\right]$ for some g.

For a given objective function, the location and nature of the optima are nonlinear qualities. The location of the global optimum for $f+g$ bears no general relationship to the location of the optimum for g or f. The error sequence has $E^{f+g}_n(z) \neq E^f_n(z) + E^g_n(z)$ for most non-constant f, g. Thus for any useful performance criterion ϕ, including the ones defined above, one expects that $\phi(\mathcal{G}, f+g) \neq \phi(\mathcal{G}, f) + \phi(\mathcal{G}, g)$ in general. Trivial parameter assignments, such as, for example, $w_n = 0$ for ϕ_w, are ignored here and elsewhere.

Performance criteria are also nonlinear in optimizers as well. For an n-dimensional cylinder set A restricting the first n coordinates of Z, the probability that A contains Z for an optimizer $\mathcal{G} + \mathcal{H}$ is given by

$$(\mathcal{G} + \mathcal{H})_f (Z \in A) = \int_{A_1} \overset{\cdots}{_n} \int_{A_n} \prod_{i=1}^{n} \mathcal{G} + \mathcal{H} \left[(Z_m)_{m=1}^{i-1}, f \right] (dx_i) . \qquad (7.12)$$

It is thus clear that $(\mathcal{G} + \mathcal{H})_f \neq \mathcal{G}_f + \mathcal{H}_f$ except under special circumstances because of the cross terms under the product. In general, $\phi(\mathcal{G} + \mathcal{H}, f) \neq \phi(\mathcal{G}, f) + \phi(\mathcal{H}, f)$.

The nonlinearity of most performance criteria has an important consequence: It opens the possibility that a convex combination over a bank of one-step optimizers may outperform any of the given optimizers. Chapter 8 will present some experimental evidence supporting this possibility, and the topic will be discussed further in Chapter 10.

7.2.2 Progressive Decomposability

Theorems 6.4.2 and 6.4.4 proved that the expected value of a random variable on the optimization process changes continuously with the objective if the value of the random variable is determined by a finite number of optimizer steps. A progressively decomposable performance criterion can be broken down into an infinite sum of finitely determined random variables.

Definition 7.2.1. *A performance criterion ϕ is progressively decomposable if there exists a sequence of functions $h_m : X^m \times \mathbb{R}^X \to \mathbb{R}$ such that*

$$\phi(\mathcal{G}, f) = \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}f} \left[h_m \left((Z_n)_{n=1}^m, f \right) \right], \qquad (7.13)$$

where $(Z_n)_{n=1}^m$ is the vector in $\mathbb{R}^m$ formed by taking the first m elements of the optimization process.

Progressive decomposability means that a performance criterion can be analyzed as the sum of infinitely many performance criteria that each depend on the state of the optimizer up to a fixed time step. This fact is used to prove that performance criteria are continuous in certain cases. Perhaps surprisingly, all of the performance criteria presented thus far are progressively decomposable.

Proposition 7.2.1. *The performance criterion ϕ_w is progressively decomposable.*

Proof. Because all terms are positive, Tonelli's theorem implies that

$$\phi_w(\mathcal{G}, f) = \sum_{m=1}^{\infty} w_m \mathbb{E}_{\mathcal{G}f} \left| f\left(Z_m^*\right) - f^* \right|, \tag{7.14}$$

which is progressively decomposable with $h_m(z,f) = w_m \left|f(z_m^*) - f^*\right|$. $\square$

Proposition 7.2.2. *The performance criteria ψ_ϵ and ψ_ϵ^N are progressively decomposable.*

Proof. Rewriting the expected value,

$$\psi_\epsilon(\mathcal{G}, f) = \sum_{m=1}^{\infty} \mathcal{G}_f\left(|f(Z_m^*) - f^*| \geq \epsilon\right), \tag{7.15}$$

which follows from $\mathcal{G}_f\left(|f(Z_m^*) - f^*| \geq \epsilon\right) = \mathcal{G}_f\left(\{\tau_\epsilon > m\}\right)$. Then

$$\mathcal{G}_f\left(|f(Z_m^*) - f^*| \geq \epsilon\right) = \mathbb{E}_{\mathcal{G}f}\left[1_{(\epsilon,\infty)}\left(E_n^*\right)\right], \tag{7.16}$$

which concludes the proof for ψ_ϵ with $h_m(z,f) = 1_{(\epsilon,\infty)}(f(z_m^*) - f^*)$. The result follows for ψ_ϵ^N by setting $h_m(z,f) = 0$ for $m > N$. $\square$

Proposition 7.2.3. *The performance criterion ζ_T is progressively decomposable.*

Proof. Without loss of generality, let $f^* = 0$. The functional ζ_T can be rewritten as follows:

$$\begin{aligned} \zeta_T(\mathcal{G}, f) &= \mathbb{E}_{\mathcal{G}f}\left[f\left(Z_\tau^*\right)\right] \\ &= \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}f}\left[f(Z_\tau^*) \mid T = m\right] \mathcal{G}_f\left(\{T = m\}\right) \\ &= \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}f}\left[f(Z_m^*) 1_{\{T=m\}}(Z)\right]. \end{aligned} \tag{7.17}$$

The final line follows because (1) $\mathcal{G}_f(\{T = m\}) = \mathbb{E}_{\mathcal{G}_f}[1_{\{T=m\}}]$, and (2) $\{T = m\}$ and $f(Z_m^*)$ are both $\mathcal{Z}_m$-measurable, since T is a stopping time. The result follows with $h_m(z, f) = f(z_m^*)1_{\{t:T(t)=m\}}(z)$. Notice that the stopping time T may depend on f without violating this result. □

Proposition 7.2.4. *The performance criteria σ_ϵ and σ_ϵ^N are progressively decomposable.*

Proof. It is possible to rewrite σ_ϵ as

$$\sigma_\epsilon(\mathcal{G}, f) = \sum_{m=1}^{\infty} \mathcal{G}_f\left(\{\tau_\epsilon = m\}\right) = \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}_f}\left[1_{\{\tau_\epsilon=m\}}\right]. \tag{7.18}$$

Since τ_ϵ is a stopping time, $\{\tau_\epsilon = m\}$ is $\mathcal{Z}_m$-measurable. Let $B_\epsilon \subseteq \mathbb{R}^m$ be given by $B_\epsilon^f = \{x \in \mathbb{R}^m : |f(x_m) - f^*| \leq \epsilon \text{ and } |f(x_k) - f^*| > \epsilon\ \forall k < m\}$. Then $h_m(z, f) = 1_{B_\epsilon^f}(z)$ makes σ_ϵ progressively decomposable. Letting $h_m = 0$ for $m \geq N$ proves that σ_ϵ^N is progressively decomposable as well. □

In fact, it is simple to prove that every performance criterion is progressively decomposable by conditioning on the natural filtration of the optimization process, $\mathcal{Z}_m$. The propositions above are still useful because they specify the decompositions h_m.

Theorem 7.2.5. *Every performance criterion as defined in Definition 7.1.2 is progressively decomposable.*

Proof. Given $h(z, f)$, let $h_1(z, f) = \mathbb{E}_{\mathcal{G}_f}[h(Z, f) \mid \mathcal{Z}_1]$ and define

$$h_m(z, f) = \mathbb{E}_{\mathcal{G}_f}[h(Z, f) \mid \mathcal{Z}_m] - \mathbb{E}_{\mathcal{G}_f}[h(Z, f) \mid \mathcal{Z}_{m-1}]. \tag{7.19}$$

Notice that $\mathbb{E}_{\mathcal{G}_f}[h_1(Z, f)] = \phi(\mathcal{G}, f)$, and for $m > 1$, $\mathbb{E}_{\mathcal{G}_f}[h_m(Z, f)] = 0$. As a result,

$$\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}_f}[h(Z, f)] = \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}_f}[h_m(z_1^m, f)] \tag{7.20}$$

□

A more restrictive property is *additive decomposability*, when the performance criterion is a linear combination of the minimal error sequence.

Definition 7.2.2. *A performance criterion* ϕ *is additively decomposable to the minimal error sequence, or just additively decomposable, if* $\phi(\mathcal{G}, f) = \sum_{m=1}^{\infty} w_m \mathbb{E}_{\mathcal{G}f}\left[f(Z^*) - f(x*)\right]$ *for some sequence* (w_m) *of nonnegative real numbers.*

Of the performance criteria presented in the last section, only ϕ_T and ζ_{T_m} are additively decomposable. Additively decomposable performance criteria induce a weaker version of No Free Lunch, introduced in Chapter 9.

7.2.3 Dependence on the Error Sequence

The performance criteria specified above all have the property that they depend primarily on the error sequence. This quality is captured by the following definition.

Definition 7.2.3. *A performance criterion* ϕ *is solely dependent on the error sequence if there is a function* $\tilde{h} : [0, \infty) \to [0, \infty)$ *such that* $\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}[\tilde{h}(E^f(Z))]$.

The performance criteria ϕ_w, ψ_ϵ, and σ_ϵ above are solely dependent on the error sequence, which can be verified by inspecting their definitions. Performance criteria that are based on stopping times, such as ζ_T and ϕ_T are not solely dependent on the error sequence in general, because the value of the stopping time may change based on factors other than the error, such as the evaluation cost along a particular trajectory.

As mentioned above, it can be important to ignore repeated evaluation points when analyzing optimizer performance. A performance criterion is *uniquely dependent* on the error sequence if it depends only on the evaluation of unique points. This property can be determined by using the unique stoping sequence (T_m) from Section 7.1.4, which yields the index of the m^{th} unique point of the optimization process. The unique stopping sequence can be used to pick out the errors at unique points.

Definition 7.2.4. *Given the error sequence* $E^f(z) = (E^f_n(z))_{n\in\mathbb{N}}$*, the unique error sequence is the subsequence of* $E^f(z)$ *determined by the unique stopping sequence* $(E^f_{T_m}(z))_{m\in\mathbb{N}}$.

Definition 7.2.5. *A performance criterion* $\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}[h(Z, f)]$ *is uniquely dependent on the error sequence, or just uniquely dependent, if* h *depends only on the unique error sequence, i.e.* $h(z, f) = \tilde{h}\left(\left(E^f_{T_m}\right)_{m\in\mathbb{N}}\right)$*. Also,* ϕ *is finitely uniquely dependent if* h *depends only on a prefix of the unique error sequence of fixed, finite length.*

Of the performance criteria above, only σ_ϵ, ζ_{T_m}, and ϕ_{T_m} are uniquely dependent on the error sequence in general. The criterion ϕ_w obviously has one term for each point including the repeated points. The criteria ψ_ϵ and ψ^N_ϵ compute the hitting time without excluding repeated points. The finite success probability σ^N_ϵ is not uniquely dependent even though σ_ϵ is because it includes repeated points to determine when N evaluations have been performed.

Each of the criteria that are not uniquely dependent can be replaced by a similar criterion that is uniquely dependent by making simple alterations. For example, the expected hitting time can be modified to $\tilde{\psi}_\epsilon = \mathbb{E}_{\mathcal{G}f}[\tau_\epsilon - R_{\tau_\epsilon}(Z)]$ where $R_m(z)$ is the number of repeated points in z up to the m^{th} component. The unique average error ϕ_{T_m} is a uniquely dependent variant of ϕ_w, and $\sigma^{T_N}_\epsilon$ is uniquely dependent as well.

Performance criteria that are finitely uniquely dependent and additively decomposable induce weak No Free Lunch theorems such as Theorem 9.3.8 under appropriate conditions, as will be seen in Chapter 9. Of the above criteria, ζ_{T_m} and ϕ_{T_m} are finitely uniquely dependent and additively decomposable whenever $T_m < \infty$.

7.3 Continuity of Performance

Continuous performance criteria are of interest because a continuous performance criterion must score an optimizer similarly on similar objective functions. The primary tools to prove the continuity of performance criteria are Theorems 6.4.2 and 6.4.4.

A performance criterion can be continuous or discontinuous in either argument. In accordance with the terminology adopted thus far, a performance criterion is continuous in objectives if small changes to the objective result in small changes to the performance. The criterion is continuous in optimizers if small changes to the optimizer do not greatly affect the performance. Continuity in objectives is examined first.

7.3.1 Continuity In Objectives

Continuity in objectives is a strong requirement, and it will not be possible to achieve it for all sequences of objectives. In this section, something slightly weaker will be proven. Given any sequence f_n such that $f_n \to f$ uniformly,[1] it will be shown that $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$ if $\mathcal{G}$ is continuous $\mathcal{G}_f$-*a.s.* The following general theorem proves that $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$ when the progressive decompositions of ϕ converge in expectation under $\mathcal{G}_f$ along a pointwise convergent sequence f_n. It will then be shown that this type of convergence follows from dependence on the error sequence when $f_n \to f$ uniformly.

Theorem 7.3.1. *Suppose ϕ is a performance criterion and $\mathcal{G} \in \mathcal{MF}$ is continuous $\mathcal{G}_f$-a.s. in objectives. Let $(f_n)_{n\in\mathbb{N}}$ be a sequence of functions converging pointwise to f. Suppose additionally that there exist functions h_m decomposing ϕ as in Equation 7.13 with the property that $\mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f_n)] \to \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)]$. Then $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$.*

Proof. First suppose $\phi(\mathcal{G}, f) < \infty$ and $\phi(\mathcal{G}, f_n) < \infty$ for all n. Fix $\epsilon > 0$. Let $f_n \to f$. Suppose without loss of generality that $f_n^* = f^* = 0$, since otherwise the function $f - f^*$ and the sequence $f_n - f_n^*$ will satisfy this equality. Theorem 7.2.5 implies that ϕ is progressively decomposable and so

$$\phi(\mathcal{G}, f) = \sum_{m=1}^{\infty} \mathbb{E}_{\mathcal{G}f}\left[h_m(Z, f)\right]. \tag{7.21}$$

Let $k_m(x) = h_m(x, f)$ and $k_{m,n}(x) = h_m(x, f_n)$ and note $k_{m,n} \to k_m$ in expectation under $\mathcal{G}f$ by the assumptions on h_m. The conditions for Theorem 6.4.2 are met for each term $\mathbb{E}\left[k_m(Z_1^m)\right]$ since $k_{m,n}$ is finitely determined and $\mathcal{G}$ is

---

[1] That is, for any $\epsilon > 0$ there is an N such that $|f_n(x) - f(x)| < \epsilon$ for $n > N$, and N does not depend on x.

continuous. Thus for appropriate $N < \infty$ and n large, the finiteness of the integrals implies that

$$\begin{aligned}
|\phi(\mathcal{G}, f) - \phi(\mathcal{G}, f_n)| &< \frac{\epsilon}{2} + \sum_{m=1}^{N} |\mathbb{E}_{\mathcal{G}f}[k_m(Z_1^m)] - \mathbb{E}_{\mathcal{G}f_n}[k_{m,n}(Z_1^m)]| \\
&\leq \frac{\epsilon}{2} + \sum_{m=1}^{N} \mathbb{E}_{\mathcal{G}f} |k_m(Z_1^m) - k_{m,n}(Z_1^m)| \\
&\quad + \sum_{m=1}^{N} |\mathbb{E}_{\mathcal{G}f}[k_{m,n}(Z_1^m)] - \mathbb{E}_{\mathcal{G}f_n}[k_{m,n}(Z_1^m)]| \\
&< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon \qquad (7.22)
\end{aligned}$$

and therefore $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$.

Next, suppose $\phi(\mathcal{G}, f) = \infty$. It must be shown that $\phi(\mathcal{G}, f_n) \to \infty$ as well. Fix $0 < M < \infty$. There is an $N < \infty$ such that

$$\phi(\mathcal{G}, f) \geq \sum_{m=1}^{N} \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)] > M. \qquad (7.23)$$

Since the sum is finite,

$$\sum_{m=1}^{N} \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)] = \mathbb{E}_{\mathcal{G}f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] > M. \qquad (7.24)$$

The integrand $\sum_{m=1}^{N} h_m(Z_1^m)$ is finitely determined. By Theorem 6.4.2,

$$\mathbb{E}_{\mathcal{G}f_n}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] \to \mathbb{E}_{\mathcal{G}f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right]. \qquad (7.25)$$

Therefore, because h_m converges in mean along f_n,

$$\mathbb{E}_{\mathcal{G}f_n}\left[\sum_{m=1}^{N} h_m(Z_1^m, f_n)\right] > \mathbb{E}_{\mathcal{G}f_n}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] - \frac{\epsilon}{2} > M - \epsilon \qquad (7.26)$$

for all n sufficiently large. Letting $\epsilon \to 0$,

$$\mathbb{E}_{\mathcal{G}f_n}\left[\sum_{m=1}^{N} h_m(Z_1^m, f_n)\right] \geq M \qquad (7.27)$$

The lower bound M was arbitrary and h_m is positive, so $\phi\left(\mathcal{G}, f_n\right) \to \infty$.

Finally, suppose that $\phi(\mathcal{G}, f_n) \to \infty$. It must be shown that $\phi(\mathcal{G}, f) = \infty$. Fix $0 < M < \infty$. There is a number K such that $\phi(\mathcal{G}, f_n) > M$ for all $n > K$. For all $n > K$, there is a number $N_0 = N_0(n)$ such that

$$\phi\left(\mathcal{G}, f_n\right) \geq \sum_{m=1}^{N_0(n)} \mathbb{E}_{\mathcal{G}f_n}\left[h_m(Z_1^m, f_n)\right] > M. \tag{7.28}$$

It is impossible that $N_0(n) \to \infty$, because this would imply $\lim_n \phi(\mathcal{G}, f_n) \leq M$. Thus $N_0(n)$ is bounded. Let N be this bound. Then for all n large,

$$\sum_{m=1}^{N} \mathbb{E}_{\mathcal{G}f_n}\left[h_m(Z_1^m, f_n)\right] > M. \tag{7.29}$$

Applying Theorem 6.4.2 again,

$$\mathbb{E}_{\mathcal{G}f_n}\left[\sum_{m=1}^{N} h_m(Z_1^m, f_n)\right] \to \mathbb{E}_{\mathcal{G}f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f_n)\right]. \tag{7.30}$$

The convergence of h_m along f_n implies

$$\mathbb{E}_{\mathcal{G}f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] > \mathbb{E}_{\mathcal{G}f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f_n)\right] - \frac{\epsilon}{2} > M - \epsilon \tag{7.31}$$

for all n sufficiently large. By taking the limit as $\epsilon \to 0$, $\phi(\mathcal{G}, f) = \infty$. □

Corollary 7.3.2. *If $f_n \to f$ uniformly and $\mathcal{G}$ is continuous $\mathcal{G}_f$-a.s. in objectives, then $\phi_w\left(\mathcal{G}, f_n\right) \to \phi_w\left(\mathcal{G}, f\right)$.*

Proof. Suppose without loss of generality that $f^* = 0$ and $f_n^* = 0$. The result will hold if $\mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f_n)] \to \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)]$. For ϕ_w, $h_m(z, f) = w_m f(z_m^*)$ under the assumptions. Because $f_n \to f$ uniformly, it follows that $h_m(z, f_n) \to h_m(z, f)$ uniformly, which proves that $\mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f_n)] \to \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)]$. The desired result follows from Theorem 7.3.1. □

The functional ζ_T is also continuous under the same conditions, provided that the stopping time T does not introduce discontinuities.

Corollary 7.3.3. *Suppose* $\mathcal{G} \in \mathcal{MF}$ *is continuous* $\mathcal{G}_f$*-a.s. in objectives, and let* $T = T_f(z)$ *be a stopping time such that* $T_{f_n}(z) \to T_f(z)$ *uniformly on a set of full* $\mathcal{G}_f$*-measure whenever* $f_n \to f$ *uniformly. Then* $\zeta_T(\mathcal{G}, f_n) \to \zeta_T(\mathcal{G}, f)$*.*

Proof. For ζ_T, $h_m(z, f) = f(z_m^*)1_{\{t:T_f(t)=m\}}(z)$. Because the stopping times are discrete, there is an N independent of z such that $T_{f_n}(z) = T_f(z)$ $\mathcal{G}_f$-*a.s.* for all $n > N$. Because $f_n \to f$ uniformly, $h_m(z, f_n) \to h_m(z, f)$ uniformly, and therefore $\mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f_n)] \to \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)]$. The result follows by applying Theorem 7.3.1. □

Corollary 7.3.3 begs the question of when T varies uniformly with the objective f. Importantly, the unique stopping times T_m are independent of the objective and therefore satisfy the assumptions of the corollary. Thus the criteria given by ζ_{T_m} converge on uniform sequences of objectives.

As another example, for a stopping time that limits the number of CPU cycles used, it seems reasonable to assume that in most cases the required number of cycles would change continuously with the objective function. There are, of course, limiting cases. For example, consider the functions

$$f_n(x) = n^{-1} \exp(-x) \sin(nx)$$

on the interval $(0, 1)$. Then $f_n \to 0$, and the zero function is trivial to compute whereas each f_n requires approximately the same time to compute on most computers. One may expect discontinuities at constant functions. In practice, however, most of the variation in computational time is due to the choice of optimization method rather than to small changes in the fitness function.

The performance criteria ψ_ϵ, ψ_ϵ^N, σ_ϵ, and σ_ϵ^N require more stringent criteria in order to prove convergence, because there exist sequences of objectives $f_n \to f$ such that $f_n - f_n^* > \epsilon$ while $f - f^* = \epsilon$. As a simple example of discontinuity, let $f_n(x) = f(x) = 0$ on $(0, 1)$, and let $f_n(x) = \epsilon + n^{-1}$ and $f(x) = \epsilon$ on $[1, 2)$. Let $\mathcal{G}$ be uniform over $(0, 2)$. Then $f_n \to f$ uniformly, but $\psi_\epsilon(\mathcal{G}, f) = 1$ and $\psi_\epsilon(\mathcal{G}, f_n) = \sum_{n=1}^{\infty} n2^{-n} = 2$. The discontinuity is caused by objectives with plateaus located at a distance of precisely ϵ away from the optimum. This problem does not arise if the trajectories with error ϵ have $\mathcal{G}_f$ measure zero.

Corollary 7.3.4. *Let $f_n \to f$ uniformly, and let $\mathcal{G} \in \mathcal{MF}$ be an optimizer that is continuous $\mathcal{G}_f$-a.s. Suppose that the set*

$$Z_\epsilon = \{z \in X^{\mathbb{N}} : |f(x_m) - f^*| = \epsilon \text{ for some m}\}$$

has $\mathcal{G}_f$-measure zero. Then $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$ when ϕ is one of ψ_ϵ, ψ_ϵ^N, σ_ϵ, or σ_ϵ^N.

Proof. On the set $X^{\mathbb{N}} \setminus Z_\epsilon$, it is not possible to have $f(z_m^*) - f^* = \epsilon$. Thus $f_n(z_m^*) - f_n^*$ must eventually be on the same side of ϵ as $f(z_m^*) - f^*$. The progressive decomposition of ψ_ϵ is $h_m(z, f) = 1_{(\epsilon,\infty)}(f(z_m^*) - f^*)$. On $X^{\mathbb{N}} \setminus Z_\epsilon$, $h_m(z, f_n) = h_m(z, f)$ for all $n > N$ with N independent of z. The progressive decomposition of σ_ϵ is $h_m(z, f) = 1_{B_\epsilon^f}(z)$ with

$$B_\epsilon^f = \{x \in \mathbb{R}^m : |f(x_m) - f^*| \leq \epsilon \text{ and } |f(x_k) - f^*| > \epsilon \ \forall k < m\}.$$

Once again, $h_m(z, f_n) = h_m(z, f)$ for all $n > N$ on $X^{\mathbb{N}} \setminus Z_\epsilon$. Thus in either case, $\mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f_n)] \to \mathbb{E}_{\mathcal{G}f}[h_m(Z_1^m, f)]$ because $\mathcal{G}_f(Z_\epsilon) = 0$, and the result follows from Theorem 7.3.1. □

So whenever $\mathcal{G}$ is continuous $\mathcal{G}_f$-*a.s.* and $f_n \to f$ uniformly, it follows that $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$ for the specific performance criteria introduced above.

7.3.2 Continuity in Optimizers

Performance criteria are continuous in optimizers everywhere, without any of the complications that arose analyzing continuity in objectives. The following theorem is analogous to Theorem 7.3.1 but with much weaker assumptions.

Theorem 7.3.5. *Every performance criterion ϕ is continuous over optimizers over all $\mathcal{MF}$.*

Proof. Let ϕ be a performance criterion. By Theorem 7.2.5, ϕ is progressively decomposable. Let $\mathcal{G}_n \to \mathcal{G}$ in $\mathcal{MF}$. Suppose without loss of generality that $f_n^* = f^* = 0$.

First, let $\phi(\mathcal{G}, f) < \infty$ and $\phi(\mathcal{G}_n, f) < \infty$. Fix $\epsilon > 0$. The finiteness and progressive decomposability of ϕ imply that there exists an $N < \infty$ such that

$$\phi\left(\mathcal{G}, f\right) < \frac{\epsilon}{2} + \sum_{n=1}^{N} \mathbb{E}\left[h_m(Z_1^m, f)\right]. \tag{7.32}$$

But now the result follows directly from Theorem 6.4.4. Setting N large,

$$\begin{aligned} \left|\phi\left(\mathcal{G}_n, f\right) - \phi\left(\mathcal{G}, f\right)\right| &< \frac{\epsilon}{2} + \sum_{m=1}^{N} \left|\mathbb{E}_{\mathcal{G}_n f}\left[h_m\left(Z_1^m, f\right)\right] - \mathbb{E}_{\mathcal{G} f}\left[h_m\left(Z_1^m, f\right)\right]\right| \\ &< \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon \end{aligned} \tag{7.33}$$

and therefore $\phi\left(\mathcal{G}_n, f\right) \to \phi\left(\mathcal{G}, f\right)$.

Next suppose that $\phi(\mathcal{G}, f) = \infty$. It must be shown that $\phi(\mathcal{G}_n, f) \to \infty$ as well. Fix $0 < M < \infty$. Then there is an $N < \infty$ such that

$$\phi\left(\mathcal{G}, f\right) \geq \sum_{m=1}^{N} \mathbb{E}_{\mathcal{G} f}\left[h_m(Z_1^m, f)\right] = \mathbb{E}_{\mathcal{G} f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] > M. \tag{7.34}$$

The integrand $\sum_{m=1}^{N} h_m\left(Z_1^m\right)$ is finitely determined. By Theorem 6.4.4,

$$\mathbb{E}_{\mathcal{G}_n f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] \to \mathbb{E}_{\mathcal{G} f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right]. \tag{7.35}$$

It follows that for n large,

$$\mathbb{E}_{\mathcal{G}_n f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] > M \tag{7.36}$$

The lower bound M was arbitrary and h_m is positive, so $\phi\left(\mathcal{G}_n, f\right) \to \infty$.

Finally, suppose that $\phi(\mathcal{G}_n, f) \to \infty$. It must be shown that $\phi(\mathcal{G}, f) = \infty$. Fix $0 < M < \infty$. There is a number K such that $\phi(\mathcal{G}_n, f) > M$ for all $n > K$. For all $n > K$, there is a number $N_0 = N_0(n)$ such that

$$\phi\left(\mathcal{G}_n, f\right) \geq \sum_{m=1}^{N_0(n)} \mathbb{E}_{\mathcal{G}_n f}\left[h_m(Z_1^m, f)\right] > M. \tag{7.37}$$

It is impossible that $N_0(n) \to \infty$, because this would imply $\lim_n \phi(\mathcal{G}, f_n) \leq M$. Thus $N_0(n)$ is bounded. Let N be this bound. Then for all n large,

$$\sum_{m=1}^{N} \mathbb{E}_{\mathcal{G}_n f}\left[h_m(Z_1^m, f)\right] > M. \tag{7.38}$$

Applying Theorem 6.4.4 again,

$$\mathbb{E}_{\mathcal{G}_n f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] \to \mathbb{E}_{\mathcal{G} f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right]. \tag{7.39}$$

Taking the limit in Equation 7.38,

$$\mathbb{E}_{\mathcal{G} f}\left[\sum_{m=1}^{N} h_m(Z_1^m, f)\right] \geq M \tag{7.40}$$

Therefore, $\phi(\mathcal{G}, f) = \infty$. □

Theorem 7.3.5 proves that every performance criterion is continuous in optimizers everywhere. Thus performance always changes smoothly as one moves from one optimizer to another along a line through $\mathcal{MF}$. Similar optimizers perform similarly on the same objective.

7.3.3 Sample Convergence and Performance Continuity

The concept of sample convergence was introduced in Chapter 5 (Definition 5.3.2). In that chapter, sample convergence was used to determine when certain optimizer convolutions are continuous. However, some optimizers are sample convergent when considered as a whole. For instance, Newton and quasi-Newton methods are sample convergent on continuously differentiable objectives, and Nelder-Mead is sample convergent on trajectories of unambivalent fitness (Definition 5.3.4). The next theorem shows that optimizers that are sample convergent $\mathcal{G}_f$-*a.s.* induce convergence of the performance criterion on under the same conditions as Theorem 7.3.1.

Theorem 7.3.6. *Suppose ϕ is a performance criterion and $\mathcal{G} \in \mathcal{MF}$ is sample convergent $\mathcal{G}_f$-a.s. in objectives. Let $(f_n)_{n\in\mathbb{N}}$ be a sequence of functions converging pointwise to f. Suppose additionally that the functions h_m in Equation 7.13 are continuous in both arguments everywhere.*

Proof. First, assume $\phi(\mathcal{G}, f) < \infty$ and $\phi(\mathcal{G}, f_n) < \infty$ for all n. It follows from Theorem 7.2.5 that ϕ is progressively decomposable. Suppose without loss of generality that $f_n^* = f^* = 0$. Fix $\epsilon > 0$. By the sample convergence of $\mathcal{G}$, the first m steps of the optimization process can only generate finitely many distinct trajectories. So there is a set of trajectories T_m^f that is finite in size such that $\mathcal{G}_f(\{Z_1^m \in T_m\}) = \mathcal{G}_f(X^{\mathbb{N}})$. Furthermore, there is a similar set $T_m^{f_n}$ for each n, and these two sets may be enumerated so that $T_m^{f_n,i} \to T_m^{f,i}$ for each i. By the definition of sample convergence, $\mathcal{G}_{f_n}(\{T_m^{f_n,i}\}) \to \mathcal{G}_f(\{T_m^{f,i}\})$ for each i since all of the trajectories are of fixed finite length. Because T_m^f and $T_m^{f,i}$ have full measure under $\mathcal{G}_f$ and $\mathcal{G}_{f_n}$, it follows from the progressive decomposability of ϕ that

$$\phi\left(\mathcal{G}, f\right) = \sum_{m=1}^{\infty} \sum_{t \in T_m^f} h_m(t, f) \mathcal{G}_f(\{Z_1^m = t\}), \tag{7.41}$$

and similarly for $\phi\left(\mathcal{G}, f_n\right)$.

As in the proof of Theorem 7.3.1, let $k_m(x) = h_m(x, f)$ and $k_{m,n}(x) = h_m(x, f_n)$ and note $k_{m,n} \to k_m$ pointwise by the assumptions on h_m. Because all of the sums are finite, for large n

$$\begin{aligned} \left|\phi\left(\mathcal{G}, f\right) - \phi\left(\mathcal{G}, f_n\right)\right| &\leq \sum_{m=1}^{\infty} \left| \sum_i k_m(T_m^{f,i}) \mathcal{G}_f(\{T_m^{f,i}\}) - k_{m,n}(T_m^{f_n,i}) \mathcal{G}_{f_n}(\{T_m^{f_n,i}\}) \right| \\ &\leq \sum_{m=1}^{\infty} \sum_i \left| k_m(T_m^{f,i}) - k_{m,n}(T_m^{f_n,i}) \right| \mathcal{G}_f(\{T_m^{f,i}\}) \\ &\quad + \sum_{m=1}^{\infty} \sum_i k_{m,n}(T_m^{f_n,i}) \left| \mathcal{G}_f(\{T_m^{f,i}\}) - \mathcal{G}_{f_n}(\{T_m^{f_n,i}\}) \right| \end{aligned} \tag{7.42}$$

Also, because $T_m^{f_n,i} \to T_m^{f,i}$, it follows that

$$\begin{aligned} \left| k_m(T_m^{f,i}) - k_{m,n}(T_m^{f_n,i}) \right| &\leq \left| k_m(T_m^{f,i}) - k_{m,n}(T_m^{f,i}) \right| + \left| k_{m,n}(T_m^{f,i}) - k_{m,n}(T_m^{f_n,i}) \right| \\ &\to 0 \end{aligned} \tag{7.43}$$

The sums on the right side of Equation 7.42 are finite, so there exists an

$N < \infty$ such that for all n sufficiently large

$$\begin{aligned}
|\phi(\mathcal{G}, f) - \phi(\mathcal{G}, f_n)| &< \frac{\epsilon}{3} + \sum_{m=1}^{N} \sum_{i} \left| k_m(T_m^{f,i}) - k_{m,n}(T_m^{f_n,i}) \right| \mathcal{G}_f(\{T_m^{f,i}\}) \\
&\quad + \sum_{m=1}^{N} \sum_{i} k_{m,n}(T_m^{f_n,i}) \left| \mathcal{G}_f(\{T_m^{f,i}\}) - \mathcal{G}_{f_n}(\{T_m^{f_n,i}\}) \right| \\
&< \frac{\epsilon}{3} + \sum_{m=1}^{N} \sum_{i} \frac{\epsilon}{3NM} + \sum_{m=1}^{N} \sum_{i} \frac{\epsilon}{3NM} \\
&\leq \epsilon, \qquad (7.44)
\end{aligned}$$

where $M = \max_{m \leq N} |T_m^f|$. Therefore $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$.

If $\phi(\mathcal{G}, f) = \infty$, then for each $M < \infty$ there is an N such that

$$\phi(\mathcal{G}, f) \geq \sum_{m=1}^{N} \sum_{t \in T_m^f} h_m(t, f) \mathcal{G}_f(\{Z_1^m = t\}) > M, \qquad (7.45)$$

and the sample convergence of $\mathcal{G}$ is sufficient to imply that each term under the same sum for $\phi(\mathcal{G}, f_n)$ converges to the term in the equation above. As a result, for any $\epsilon > 0$

$$\phi(\mathcal{G}, f_n) \geq \sum_{m=1}^{N} \sum_{t \in T_m^{f_n}} h_m(t, f_n) \mathcal{G}_f(\{Z_1^m = t\}) > M - \epsilon, \qquad (7.46)$$

similar to Equation 7.26 in Theorem 7.3.1. Taking the limit as ϵ goes to zero and observing that M is arbitrary together imply that $\phi(\mathcal{G}, f_n) \to \infty$.

The final case, $\phi(\mathcal{G}, f_n) \to \infty \implies \phi(\mathcal{G}, f) = \infty$, can be proven by extending the final case in the proof of Theorem 7.3.1 in analogy with the prior paragraph. $\square$

Theorem 7.3.6 is stronger than Theorem 7.3.1 because it proves that most performance criteria are continuous in objectives without requiring the objective to converge uniformly. In particular, each of the corollaries in Section 7.3.1 has an analogue for sample convergent optimizers that is the same in all respects, except that the sequence f_n only needs to converge to f pointwise rather than uniformly.

Since the vast majority of optimizers are either sample convergent or almost surely continuous on most objectives, Theorems 7.3.6 and 7.3.1 together imply that standard measures of performance are generally continuous on all optimizers and most objectives.

7.4 Conclusion

This chapter introduced a flexible framework for analyzing performance criteria for optimizers. Specific categories of performance criteria were presented, most of which correspond to the experimental quantities that are commonly reported in the literature. All performance criteria were shown to be progressively decomposable into sums of finite expectations, and this fact was leveraged to prove that most performance criteria are continuous subject to certain conditions.

Up to this point, the properties of performance criteria have been discussed in the abstract, but the value of these performance criteria can also be measured experimentally, which is done next in Chapter 8. The experiments in that chapter will demonstrate concretely the features of the performance criteria introduced in this chapter, including continuity and convergence. More specifically, they demonstrate that in practical terms, certain optimizers appear to perform better than others on problems of interest using the performance criteria defined here.

After that experimental interlude, Chapter 9 will study the important theoretical question of whether some optimizers are better than others, proving for the first time the exact conditions under which all optimizers have equivalent performance. These proofs will rely heavily on the unique stopping sequence and the concept of unique dependence on the error sequence. The implication of the No Free Lunch Identification Theorem 9.3.7 is that performance is only equivalent in settings where learning is impossible.

Chapter 8

Performance Experiments

Several performance criteria were defined and analyzed in Chapter 7. This chapter reports the results of experiments run to estimate the values of these performance criteria on a bank of standard optimizers and objectives. The complete results are provided for reference in tabular form in Appendix A. They are summarized and discussed below. In addition, the theoretical continuity of performance criteria is illustrated through several examples. The final section of the chapter applies principal components analysis to the experimental performance values in order to visualize the position of the standard optimizer set in the space of long-running optimizers.

8.1 Experimental Setup

Experiments were performed in real-vector space on a set of twelve standard benchmarks. The search space was $X = \mathbb{R}^d$ with the topology induced by the standard Euclidean metric, $d(x, y) = \sum_{i=1}^{d} |x_i - y_i|^2$. The benchmarks were optimized within a benchmark-specific hypercube $Q \subseteq \mathbb{R}^d$, and feasibility regions were used to prevent the optimizers from escaping the constraints. That is, for each benchmark f, the experiment was performed with an altered objective $\tilde{f}$ given by

$$\tilde{f}(x) = \left\{ \begin{array}{cc} f(x) & x \in Q_f \\ \infty & \text{otherwise} \end{array} \right. , \tag{8.1}$$

where Q_f is the hypercube constraining f.

8.1.1 Benchmarks

The twelve benchmarks are defined in Table 8.1. These benchmarks are commonly used to test global optimizers. They cover a broad cross-section of possible objective functions, including objectives that are convex, multimodal, periodic, differentiable, nowhere differentiable, deceptive, and irregularly shaped. All of the optimizers tested were continuous and bounded. The definitions and descriptions of these benchmarks can be found in the literature [2, 7, 24, 56, 139, 212]. Notably, two versions of Ackley's function exist; both are included in the comparisons. The less common one is termed *log-ackley* and is due to [2]. The more common version of the benchmark is exponentiated and centered and is simply termed *ackley*.

Each benchmark was tested in five, ten, and 25 dimensions ($d = 5, 10, 25$), except that *shekel* and *langerman* were tested in five and ten dimensions only (since they are not defined in 25 dimensions). The feasible region for each benchmark was a bounded hypercube with the range for each component shown in the table. The minima for these functions are known, as shown in Table 8.1 for five dimensions with precision up to 10^{-4}. The actual values are known up to machine-level precision (10^{-16}), and these more accurate

Table 8.1: Benchmarks for Experimental Validation with dimension $d = 5, 10, 25$. Minimum for $d = 5$.

Name	Definition	Minimum	Domain
sphere	$\sum_{i=1}^{d} x_i^2$	0.0000	(-5.12 , 5.12)
ackley	$-20\exp(-\frac{.02}{d}\|\|x\|\|^2) - \exp(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)) + 20 + e$	0.0000	(-30 , 30)
log-ackley	$\sum_{i=1}^{d-1} e^{-0.2}\sqrt{x_i^2 + x_{i+1}^2} + 3\cos(2x_i) + 3\sin(2x_{i+1})$	-13.3796	(-30 , 30)
whitley	$\sum_{i=1}^{d}\sum_{j=1}^{d}\frac{w(x_i,x_j)^2}{4000} - \cos(w(x_i, x_j)) + 1$, with $w(y,z) = 100\left(y^2 - z\right)^2 + (1-z)^2$	0.0000	(-30 , 30)
shekel	$\sum_{i=1}^{30}\frac{1}{\sum_{j=1}^{d}(x_j - a_{ij})^2 - c_i}$	-10.4056	(-5 , 15)
rosenbrock	$\sum_{i=1}^{d-1} 100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2$	0.0000	(-5.12 , 5.12)
rastrigin	$10d + \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i)$	0.0000	(-5.12 , 5.12)
salomon	$-\cos(2\pi\|x\|) + 0.1\|x\| + 1,\ \|x\| \equiv \left(\sum_i x_i^2\right)^{1/2}$	0.0000	(-30 , 30)
langerman	$-\sum_{i=1}^{5} c_i \exp(-y_i/\pi)\cos(\pi y_i),\ y_i = \sum_{j=1}^{d}\left(x_j - a_{ij}\right)^2$	-0.9650	(-5 , 15)
schwefel	$d^{-1}\sum_{i=0}^{d} -x_i \sin\sqrt{\|x_i\|}$	-418.9829	(-512 , 512)
griewank	$1 + \sum_{i=1}^{d}\frac{x_i^2}{4000} - \prod_i \cos(x_i/\sqrt{i})$	0.0000	(-600 , 600)
weierstrass	$\sum_{i=1}^{d}\sum_{j=1}^{20} 0.5^j \cos\left(2 \cdot 3^j \pi(x_i + 0.5)\right) + d\sum_{j=1}^{20} 0.5^j \cos\left(3^j \pi\right)$	0.0000	(-0.5 , 0.5)

values were used for testing the accuracy of the experiments. Heat maps of the eleven benchmarks with $d = 2$ are shown in Figure 8.1.

8.1.2 Algorithms

The performance of nine algorithms was measured. These algorithms were (1) conjugate gradient descent (CG), (2) Nelder-Mead (NM), (3) a generating set search (GSS), (4) simulated annealing (SA), (5) a real-coded genetic algorithm (rGA), (6) an evolution strategy (CMA-ES), (7) differential evolution (DE), (8) particle swarm optimization (PSO), and (9) the real-coded Bayesian Optimization Algorithm (rBOA). As discussed in Chapter 2, these algorithms cover a broad spectrum of stochastic optimization algorithms and represent a general sampling of the current state of the art. They are known to be effective on a wide array of fitness functions and most of them perform reasonably well on the selected benchmarks. For all of the algorithms, parameters were set according to the literature where available and hand-tuned otherwise to optimize performance.

Conjugate gradient descent with estimated gradients was tested using the publicly available fmin_cg implementation from the SciPy package with its defaults. The Nelder-Mead algorithm was described in Section 2.4.1 and was implemented in the standard form. GSS was based on the direct search algorithm described in Section 2.4.2 using the positive spanning set of size $d + 1$ and no search heuristic.

Simulated annealing was run as a single chain with a logarithmic cooling schedule (Section 2.5.1). It was restarted randomly with probability 0.001 after each point. The rGA method was a standard real-coded genetic algorithm using linear ranking selection with pressure 1.8, uniform crossover, and Gaussian mutation (Equation 4.21). The mutation variance for rGA was set to 0.05 for all problems except *schwefel* and *griewank*, where it was set to 10.

CMA-ES is the Correlated Matrix Adaption algorithm of Hansen and Ostermeier (Section 2.7.3) and was tested with four different population sizes: 100, 750, 1250, and 2500 [84] . At each generation, 50% of the population was used to build an updated normal distribution.

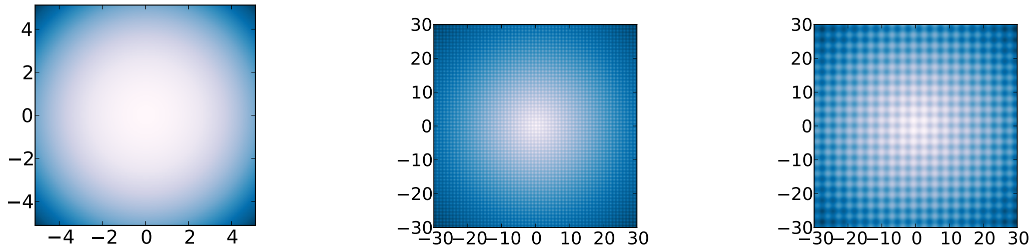
DE [198] was trained with four different parameter settings, one each with crossover rates 0.2 and 0.9 and learning rates 0.2 and 0.9 (Section 2.7.1). PSO [62] was trained with both the global and local adaptation rates set to 2.0 (Section 4.3.1). The velocity decay was tested with two different values, -0.5 and 1.0, following results by Pedersen [153] on optimal parameter settings for PSO. The rBOA method is an Estimation of Distribution Algorithm (EDA), a class of optimizers introduced in Section 2.7.2. It was implemented as described by Ahn et al. in [3].

Many optimizers converge quickly to a local optimum, and restarting an optimizer can be an effective strategy to bootstrap its performance. To demonstrate this idea, CG, NM, GSS, and CMA-ES were restarted on convergence to improve performance. Results of this nature have been reported previously in the literature for CMA-ES [13]. The restarted versions are referred to as CG-R, NM-R, GSS-R, and CMA-ES-R, respectively. Other methods could also benefit from restarting, but these four methods should benefit most, since they converge quickly.

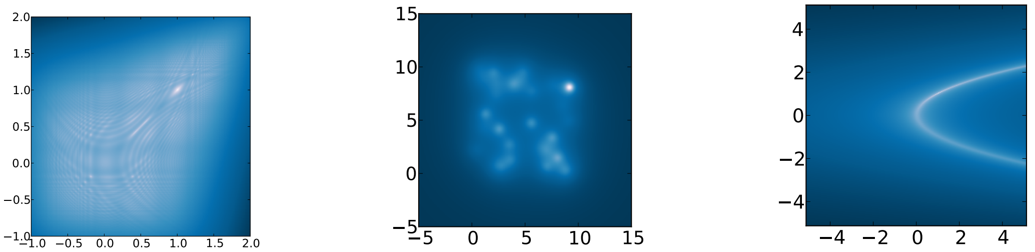
All algorithms were run on all benchmarks 200 times for each tested parameter setting. These 200 runs are sufficient to guarantee statistical significance on the estimated success rates σ_ϵ^N for each algorithm at the 95% level within ±0.5% [204]. The variance on other performance criteria was large, but not deleteriously so (see Figures 8.8 and 8.9 for visual examples and the tables in Appendix A for exact numbers). When a single number is shown as the result of an experiment, that number represents the best value achieved on any parameter setting for that algorithm, unless otherwise stated. Experiments with different parameters are shown separately in Appendix A.

8.1.3 Scaling Factors

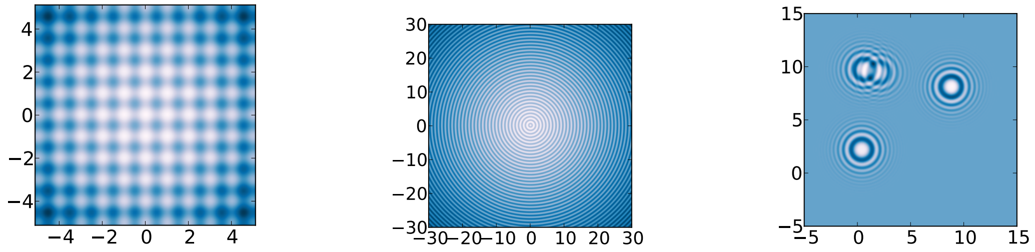
The experimental results contain estimates of the error of each algorithm on the benchmarks. The performance criteria ζ_T and ϕ_w are computed from this error. Because the magnitude of the error depends on the internal scaling of each objective function, comparisons across benchmarks are not numerically meaningful without scaling. For example, by multiplying the scaled error values in Table A.15 by the scaling factors in Table A.1, it would



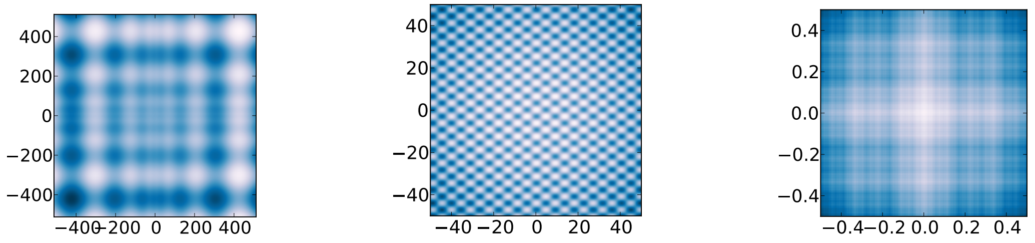
(a) sphere (b) ackley (c) log-ackley



(d) whitley (e) shekel (f) rosenbrock



(g) rastrigin (h) salomon (i) langerman



(j) schwefel (k) griewank (l) weierstrass

Figure 8.1: Heat maps for the twelve benchmark functions in two dimensions ($d = 2$). The benchmarks *whitley* and *griewank* are scaled to show the critical region. These benchmarks include unimodal, multimodal, periodic, irregular, and discontinuous functions, resulting in a broad test of an optimizer's capabilities

seem that rGA has its worst performance on *schwefel* out of all the benchmarks. However, comparing the performance of all optimizers on *schwefel*, rGA does better than all but three other optimizers, two of which benefitted from restarts. Thus scaling is necessary.

Scaling factors were computed by estimating the performance of random search on each benchmark. To this end, $10,000$ points were sampled uniformly from the bounding cube for each benchmark objective, and the minimum error from the optimum was recorded. This procedure was repeated 100 times and the results were averaged. The scaling factors computed in this way were only computed once for each dimension and were reused throughout dissertation. They are listed for each benchmark in Table A.1.

The scaling factors used in these experiments reveal the ratio of each algorithms performance to the performance of random search. There are other ways that scaling could have been accomplished. For example, the norm of each objective could have been estimated as the scaling factor, but it is not always easy to obtain a practical estimate of $||f||$. More importantly, the objective was not assumed to be integrable, although each one of these benchmarks can be integrated on the search domain. Since random search is a suitable comparison point for analyzing optimizer performance, the scaling factors that were used are meaningful and do make it possible to compare the performance of a single optimizer across several benchmarks.

8.2 Experimental Results

Figures 8.2 to 8.6 provide a visualization of the performance criteria introduced in Section 7.1 for the benchmarks in five dimensions; the complete experimental results are given in Appendix A. The performance criteria in the figures group the evaluations into virtual populations of 100 each so that the experiment contains $2,500$ successive populations. Population-based optimizers with larger populations and optimizers that do not use populations are thus compared in the same setting.

Figure 8.2 displays a scaled instance of ϕ_w with weights $w_{100n} = \frac{1}{2500}$

for $10 \leq n \leq 2500$ and zero otherwise. That is,

$$\phi_1\left(\mathcal{G}, f\right) = \frac{1}{s_f}\frac{1}{2490}\sum_{n=10}^{2500} \mathbb{E}_{\mathcal{G}f}\left[f(Z^*_{100n}) - f^*\right], \tag{8.2}$$

where s_f is the objective-specific scaling factor from Table A.1. The initial factor of $\frac{1}{2490}$ was used to scale the magnitude of the sum, and the sum was started at $n = 10$ in order to ignore the initial error of the first $1,000$ evaluations. This bar chart provides a sense of how different optimizers compare to each other on each objective. DE, CMA-ES, CMA-ES-R, GSS-R, and NM-R perform best on this criterion. As expected, CG and even CG-R perform poorly overall on this benchmark set.

Figure 8.3 displays another scaled instance of ϕ_w, this time with exponential decay,

$$\phi_2\left(\mathcal{G}, f\right) = \frac{1}{s_f}\sum_{n=10}^{2500} \frac{1}{2^{n-10}}\mathbb{E}_{\mathcal{G}f}\left[f(Z^*_{100n}) - f^*\right]. \tag{8.3}$$

Once again, the sum was started after $1,000$ evaluations to avoid early errors. Unlike ϕ_1, ϕ_2 places higher emphasis on early errors. Thus by comparing Figure 8.3 with Figure 8.2 it is possible to obtain a sense of the convergence speed of each optimizer on the benchmarks. The values of ϕ_2 are larger than ϕ_1 for most optimizers, reflecting the earlier errors. Importantly, this effect is less pronounced in CMA-ES, GSS, and NM, which converge faster than the other algorithms. The term "converge" here is intended to mean "cease to propose substantially new evaluation points" rather than "converge to an optimum", although it is known that each of the algorithms mentioned do converge to a local optimum. The restarted versions of these three algorithms do perform worse when using ϕ_2 rather than ϕ_1 because restarting lowers the average error substantially in later evaluations, which are less important under ϕ_2.

The scaled values for $\zeta_{T_{250,000}}$, the minimum global error at the $250,000^{th}$ unique evaluation, are shown in Figure 8.4. Whereas ϕ_1 and ϕ_2 give the average error under different weightings, $\zeta_{T_{250,000}}$ gives the error at the final evaluation. In $\mathbb{R}^m$, non-unique points have measure zero under the selected optimizers, so $T_{250,000} = 250,000$ on these experiments. The criterion ζ_T gives

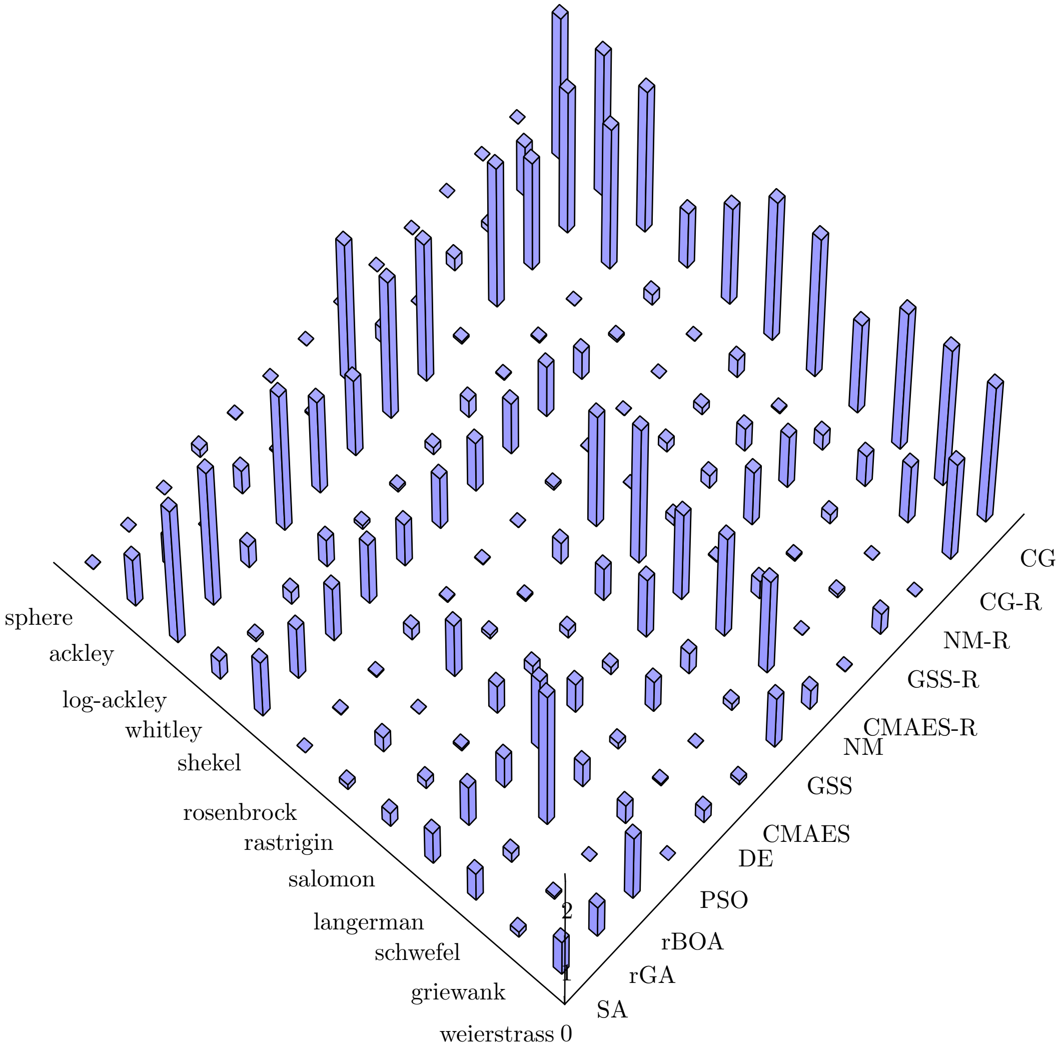


Figure 8.2: Performance values for selected optimizers on the twelve benchmarks using the performance criterion ϕ_1, which averages global error over $250,000$ evaluations, starting after $10,000$ evaluations. Lower values are better. DE, CMA-ES, CMA-ES-R, GSS-R, and NM-R perform best on this performance criterion.

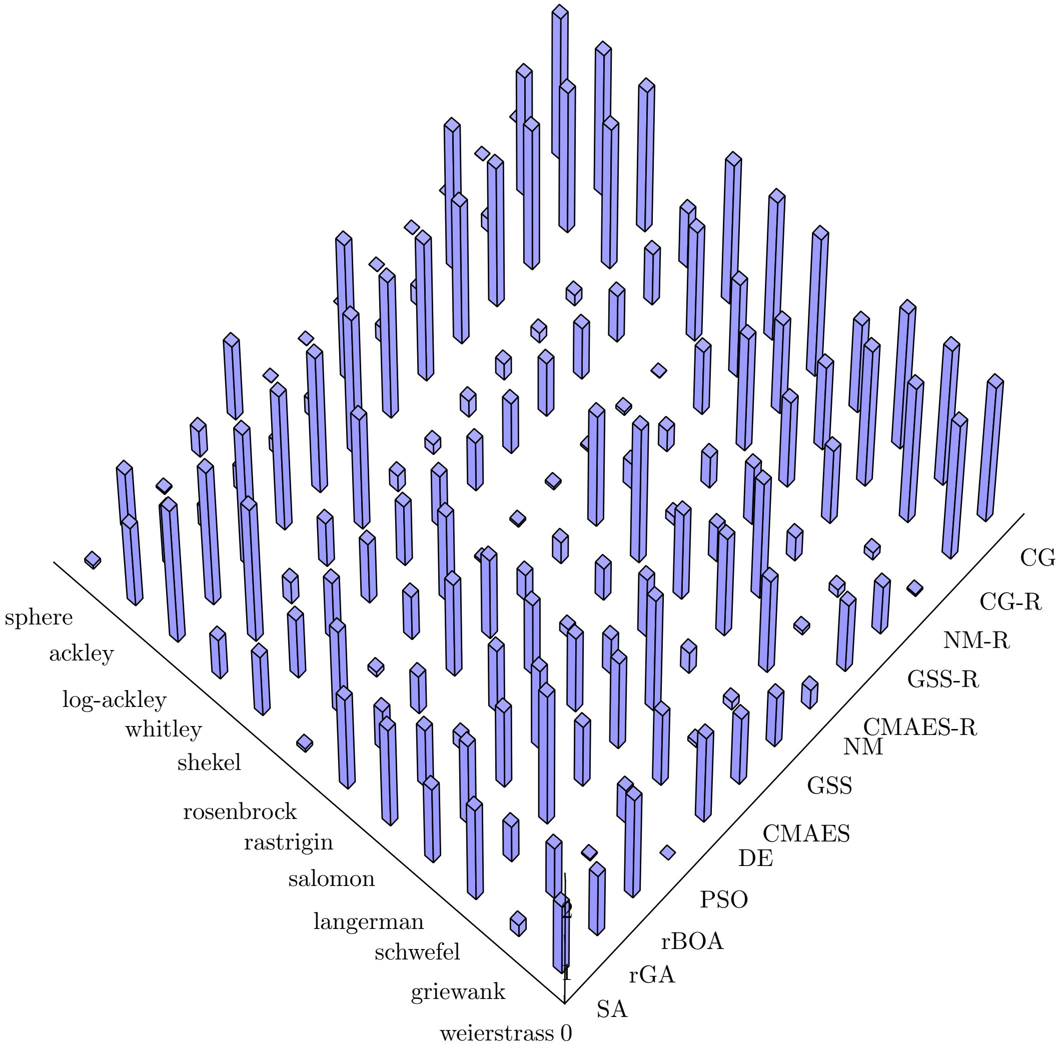


Figure 8.3: Performance values for selected optimizers on the twelve benchmarks using the performance criterion ϕ_2, which sums global error with an exponential decay over $250,000$ evaluations, starting after $10,000$ evaluations. Lower values are better. Unlike ϕ_1, ϕ_2 counts earlier errors more heavily, and thus prefers optimizers that converge faster, such as NM, GSS, and CMA-ES.

little information about the speed of convergence but is useful for comparing the absolute performance of different optimizers on a fixed objective. In Figure 8.4, DE is most reliable algorithm, with CMA-ES not far behind. This conclusion can also be drawn from the results for ϕ_1, but is less clear. In ϕ_2, the fast-converging optimizers appear preferable – especially CMA-ES and GSS. Restarting improves performance, with CMA-ES-R, GSS-R, NM-R, and CG-R all performing well on ζ_T. On ζ_T, at least, DE still appears preferable to the restarted optimizers.

Comparing the different criteria reveals a tradeoff between solution quality and convergence speed. DE achieves solution quality by exploring the space more thoroughly. CMA-ES provides slightly worse solution quality in much faster time. The restarted algorithms also converge slower but achieve higher quality. The desired tradeoff can be achieved to some extent by choosing the appropriate algorithm.

The average hitting time ψ_ϵ^N represents the convergence time directly. This performance criterion is displayed with $\epsilon = 0.01$ and $N = 250,000$ in Figure 8.5, scaled to represent the number of virtual populations of size 100 before the hitting time. The values are noisy and generally quite large due to the high variability of this criterion. Overall, it is difficult to draw conclusions from Figure 8.5, and the success-only hitting time $\hat{\psi}_\epsilon^N$ is preferable. Since only a percentage of the trial runs hit the error threshold, each average includes a substantial number of copies of the maximum, N. Consistently low values are only achieved when the optimizer converges on almost every run, as PSO does on *weierstrass*. The criteria ψ_ϵ^N, $\hat{\psi}_\epsilon^N$, and σ_ϵ^N are somewhat sensitive to the scaling of the objective function, but their values have a much more consistent meaning when comparing the performance of an optimizer on different objectives, so scaling was not applied to the error threshold.

A clearer picture of the convergence speed is given by Figure 8.6, which shows the criterion $\hat{\psi}_\epsilon^N$ from Equation 7.8 with the same parameters. This criterion lacks the high variability of ψ_ϵ^N. Optimizers that universally failed to attain the error target are shown with values of $2,500$ in Figure 8.6. This figure shows that CG, CMA-ES, GSS, NM, and rBOA are the optimizers with the fastest convergence speeds, although each of them fail on some subset of

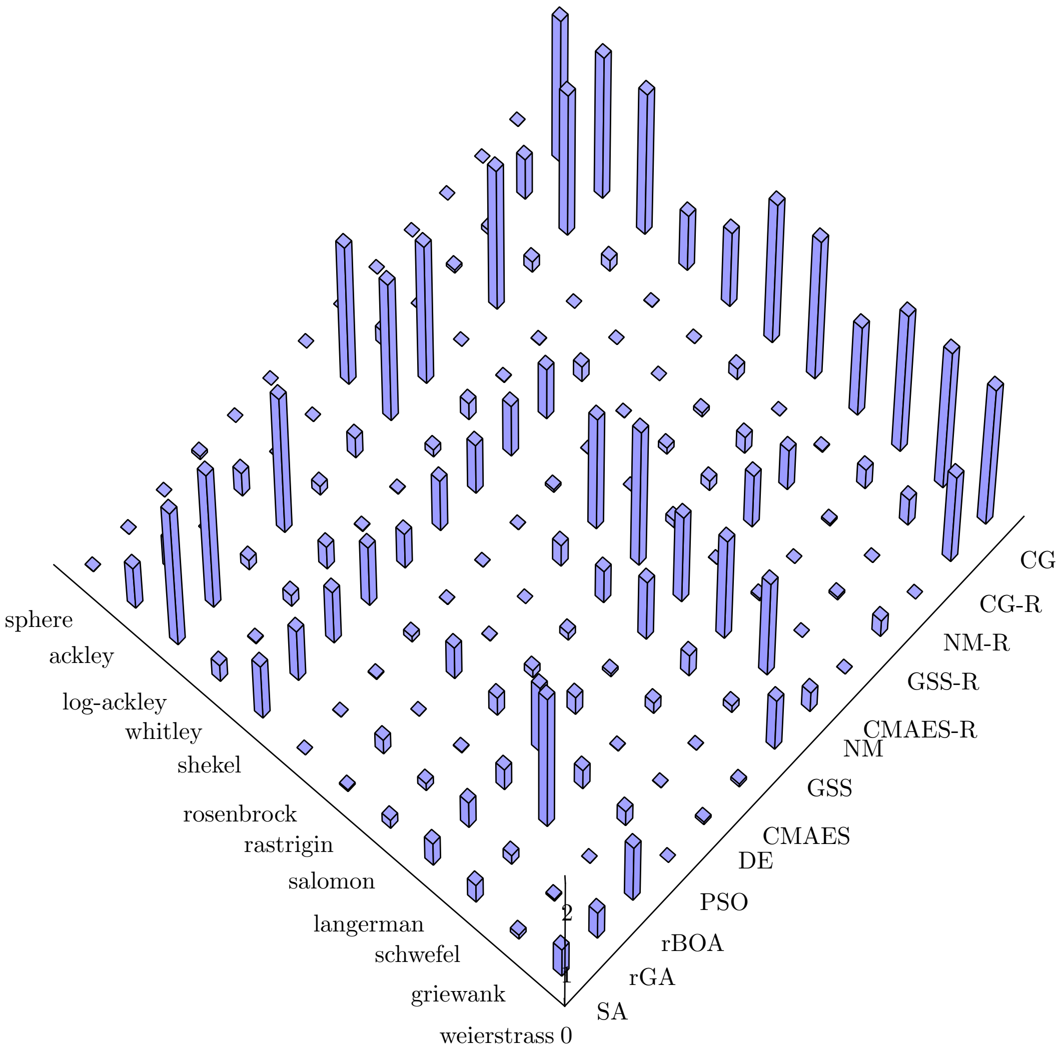


Figure 8.4: Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\zeta_{T_{250,000}}$, which reports the minimum global error after $250,000$ unique evaluations. Lower values are better. Because it does not sum over multiple time steps, ζ_T communicates little information about the convergence rate. DE performs best among the optimizers, with CMA-ES close behind. Restarting improves performance with enough evaluations, and so CMA-ES-R, GSS-R, NM-R, and CG-R each perform well.

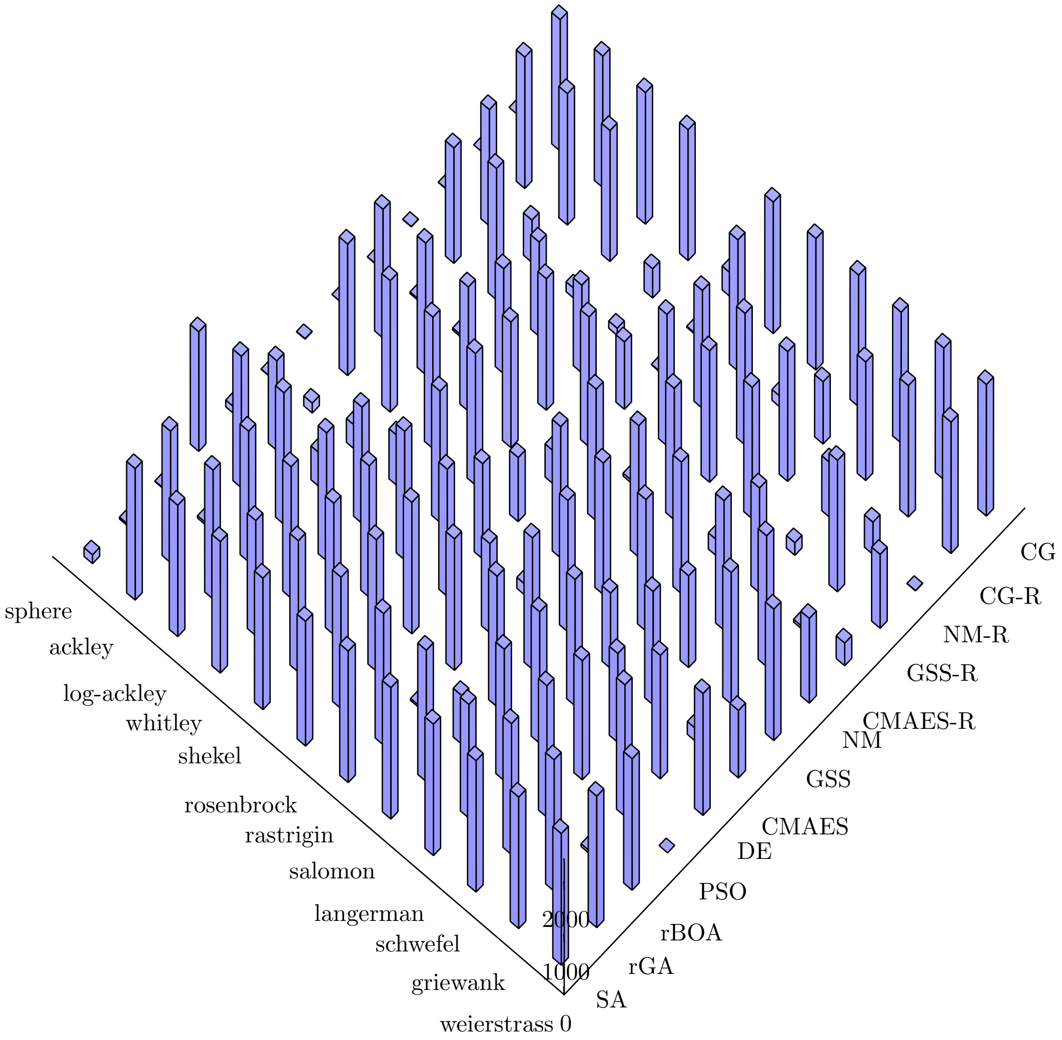


Figure 8.5: Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\frac{1}{100}\psi_\epsilon^N$ with $\epsilon = 0.01$ and $N = 250,000$. This criterion records the average number of evaluations before the minimum global error drops below ϵ, capped at a maximum of N. Lower values are better. Scale as shown runs from zero to $2,500$ and represents the number of generations until the hitting time with a notional population size of 100. Because many trial runs fail on these benchmarks, the numbers are typically high, and often near $2,500$. Very low values, as seen for *sphere*, indicate fast convergence. In general, the success-only hitting time $\hat{\psi}_\epsilon^N$ is more preferable than ψ_ϵ^N for measuring performance.

the benchmarks. Restarting, which improves performance on ζ_T, predictably weakens performance on $\hat{\psi}^N_\epsilon$.

The success probability σ^N_ϵ complements the values of $\hat{\psi}^N_\epsilon$. It is shown in Figure 8.7. Unlike in the other figures, higher values of σ^N_ϵ indicate higher probability of success and thus larger bars are better. Figure 8.7 shows which optimizers are the most reliable overall. The restarted optimizers have the highest success probabilities, with some lapses. Among the optimizers that do not restart, DE and CMA-ES are the most reliable and consistent at reaching the error target, with CMA-ES appearing preferable in this figure. Comparing with the values for ζ_T in Figure 8.4, it can be surmised that when DE fails to reach the error target ϵ, it still attains a local minimum close in value to the true global minimum, whereas CMA-ES makes larger errors when the error target is not attained. Thus, the performance criteria that use hitting times ignore catastrophic failures in favor of frequent successes, and CMA-ES appears more reliable than DE in this regard.

Overall, the choice of performance criterion should reflect the preferences of the practitioner, balancing tradeoffs of convergence speed, solution quality, and consistency. If both solution quality and speed are important, then the pair $(\hat{\psi}^N_\epsilon, \sigma^N_\epsilon)$ is a good choice. In this case, the three restarted algorithms performed best on the benchmarks. If solution quality and consistency are paramount but speed is less of a concern, then ζ_T is the best choice. To balance convergence speed as well, ϕ_1 can be used. In either case, DE appears to be the most reliable non-restarted optimizer. Its performance could be further bootstrapped by restarting as well. The value of restarting appears as a constant theme through these experiments. This topic is discussed more thoroughly in the next section.

8.3 Restarting to Bootstrap Performance

The restarted algorithms were undeniably the best performers out of all the optimizers on nearly every performance criterion and benchmark. Given that high number of evaluations performed $(250,000)$, it is perhaps not surprising that restarting would have a beneficial effect. Most of the optimizers tested tend to converge quickly to a small region of the search space, which

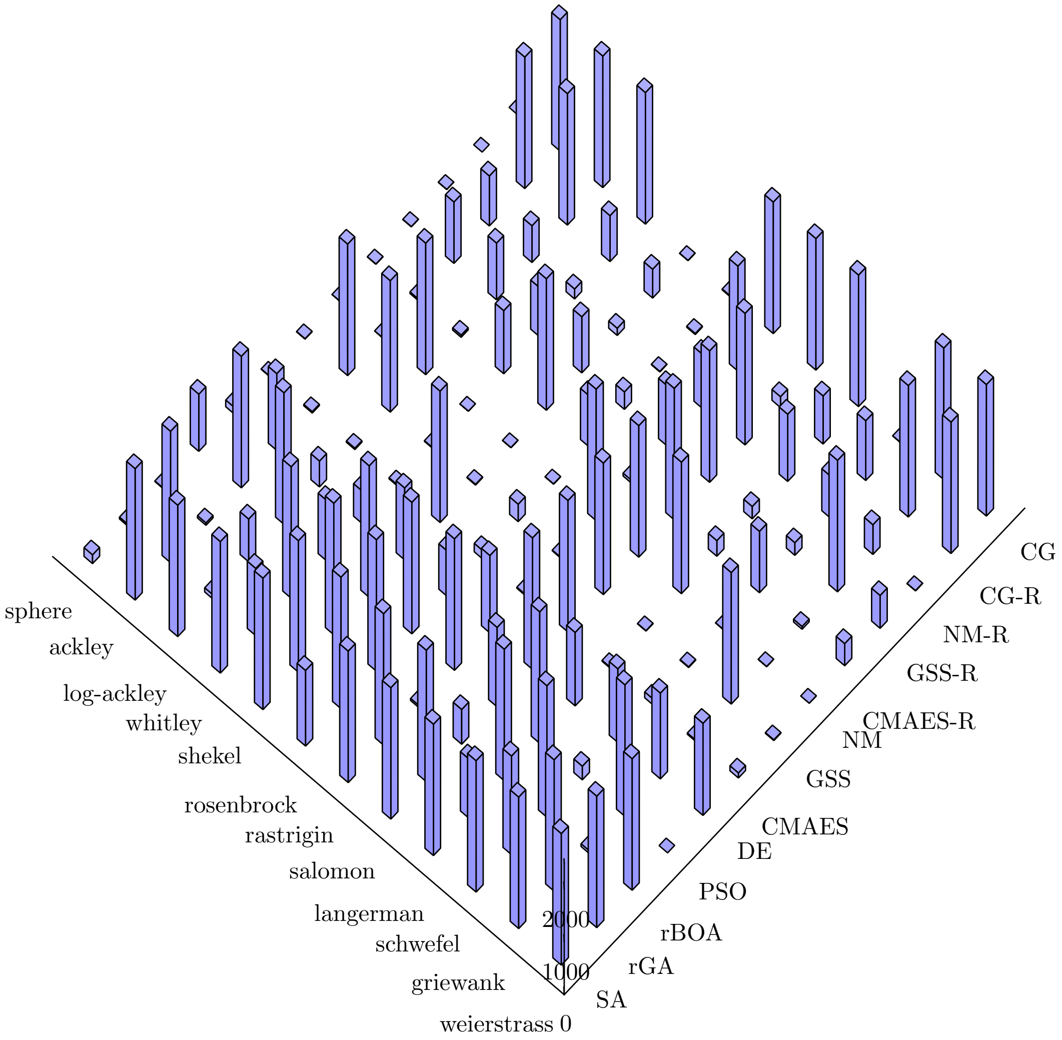


Figure 8.6: Performance values for selected optimizers on the twelve benchmarks using the performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.01$ and $N = 250,000$ (see Equation 7.8). This criterion records the average number of evaluations before the minimum global error drops below ϵ on trial runs where this error is attained. If the error threshold is never attained, the value is set at N. Lower values are better. Scale as shown runs from zero to $2,500$ and represents the number of generations until the hitting time with a notional population size of 100. These values give a clear picture of the relative convergence speed of various optimizers. For example, on successful trials, CMA-ES, GSS, and NM converge very quickly, whereas DE converges, but more slowly.

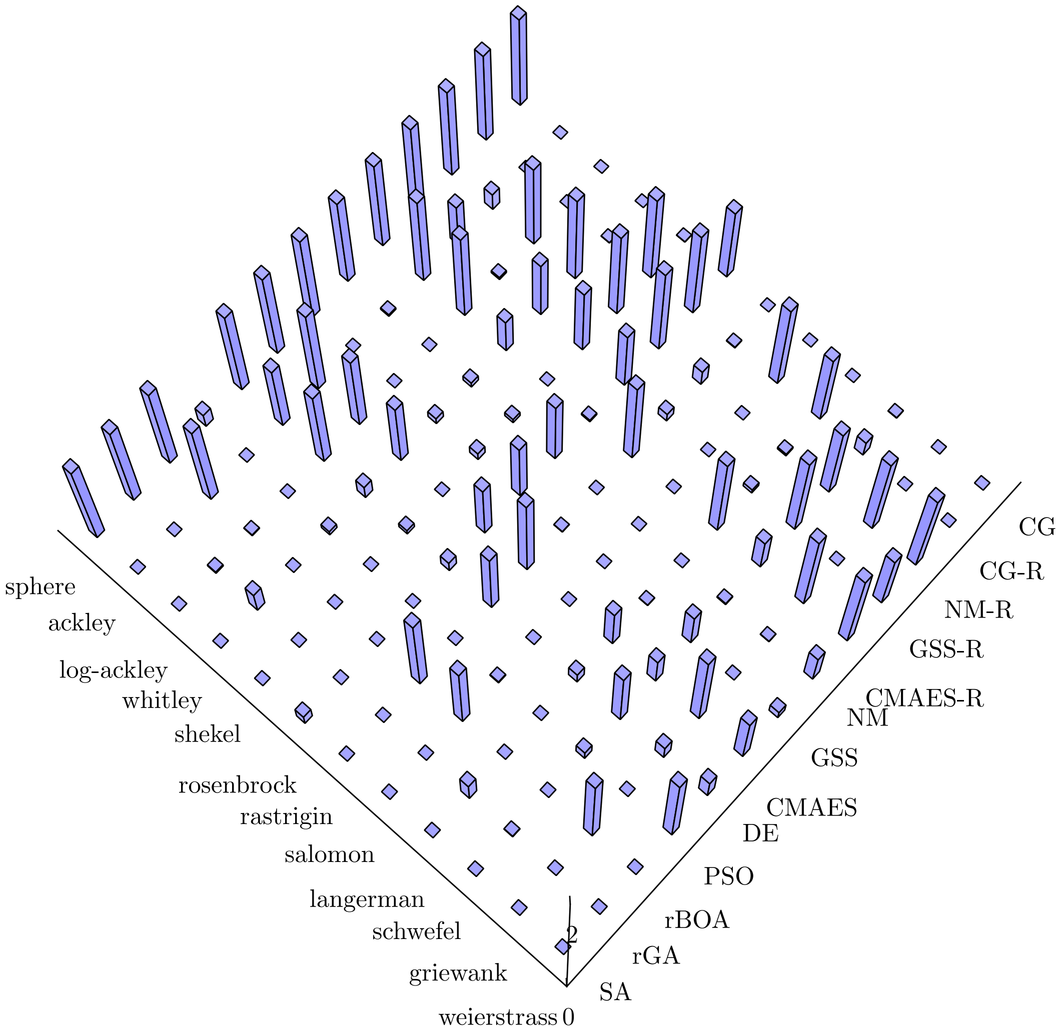


Figure 8.7: Performance values for selected optimizers on the twelve benchmarks using the performance criterion σ_{ϵ}^{N} with $\epsilon = 0.01$ and $N = 250,000$. This criterion computes the probability of attaining global error within ϵ of the true optimum. Values run from zero to one; higher values are better. Overall, CMA-ES (especially CMA-ES-R) performs best on this performance criterion. DE, GSS-R, and NM-R also perform well on different objectives.

they then sample in increasing detail. This convergence is faster in lower dimensions, as can be seen from the tables in Appendix A. In 25 dimensions, the restarted optimizers are less distinguishable from their non-restarted versions. The reason is that in higher dimensions, the optimizers converge more slowly, and thus are restarted less often. If the experiments were run for substantially more evaluations, then the beneficial effect of restarting might reappear even in higher dimensions.

It is not necessary to wait for convergence to restart an optimizer, and there may be a benefit to restarting after a fixed number of evaluations. To observe the effect mathematically, suppose an optimizer has a success probability of $\sigma_\epsilon^{25000} = 0.05$ on a particular objective after $25,000$ evaluations. If the optimizer is restarted every $25,000$ evaluations, then after $250,000$, the optimizer will have been run 10 times, and its success probability can be calculated. This value may be computed recursively as the sum of $p_n = p_{n-1} + (1 - p_{n-1}) \times p_1$ for $1 \leq n \leq 10$ with $p_1 = 0.05$. In this case, $\sigma_\epsilon^{250000} = 0.40$. If the same success probability can be achieved with $10,000$ evaluations, then $\sigma_\epsilon^{250000} = 0.72$ will be reached. In this way, fast but infrequent convergence can be parlayed into reliable but potentially slow convergence. There is always a tradeoff between speed and quality, but it is possible to improve both with effective restarts.

8.4 Illustrating Continuity of Performance

In Chapter 7, substantial effort was expended to demonstrate the continuity of performance criteria as optimizers and objectives are changed. These results suggested that simplified approximations to an optimizer or an objective can be used to predict the performance of a similar but more complex optimizer. This result is both practical and important, and it is worthwhile to demonstrate such continuity graphically. This section includes results that illustrate three facts proven in the Chapter 7. First, the performance of continuous or sample convergent algorithms changes continuously with the objective. Second, the performance of similar optimizers is similar on the same problems. Third, when the conditions of the theorems in Chapter 7 are not met, discontinuities may be encountered.

8.4.1 Continuity in Objectives

In Section 7.3.1, it was proven that $\phi(\mathcal{G}, f_n) \to \phi(\mathcal{G}, f)$ if $f_n \to f$ subject to certain conditions. This section looks at how the performance changes for a fixed optimizer as the objective changes. For this purpose, the benchmark objectives *shekel* and *langerman* were convexly combined to form a line in objective space given by

$$f_\alpha(x) = \alpha \, \mathit{langerman}(x) \, + \, (1-\alpha) \, \mathit{shekel}(x). \tag{8.4}$$

The optimizer NM-R was run for 200 trials on a range of objectives f_α with $\alpha = 0.0, 0.05, 0.10, \ldots, 0.95, 1.0$. The results are shown for different performance criteria in Figure 8.8. In these experiments, it is difficult to know the minimum f^*_α exactly, and so the best observed value on any trial was taken as the minimum. As long as the true success probability is positive for one of the two optimizers, it is reasonable to estimate the minimum in this way.

As Figure 8.8 shows, the performance changes smoothly as α runs from zero to one on four different performance criteria: ϕ_1, ζ_T, $\hat{\psi}^N_\epsilon$, and σ^N_ϵ. Lines indicating the first standard deviation are shown, with the performance value in bold. The variance cannot be computed from these experiments for σ^N_ϵ, but should be less than 0.005 with high probability. Referring to Figure 8.7, it can be seen that NM-R succeeds frequently on *shekel* but rarely on *langerman*. The smooth and nonlinear transition in performance values as α runs from zero to one is expected, since NM-R is sample convergent $\mathcal{G}_f$-*a.s.* on objectives without plateaus. In this situation, Theorem 7.3.6 implies

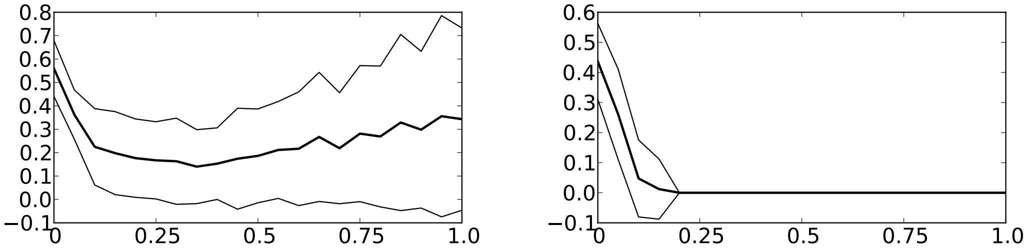
$$\phi(\mathrm{NM-R}, f_{\alpha_n}) \to \phi(\mathrm{NM-R}, f_\alpha)$$

whenever $\alpha_n \to \alpha$. The experiments are thus in line with the theory.

8.4.2 Continuity in Optimizers

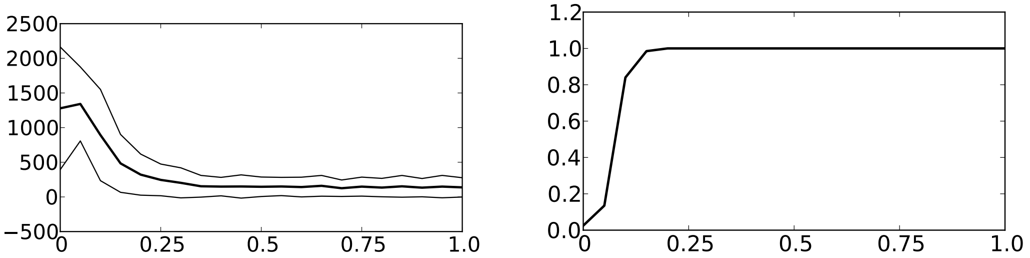
In Section 7.3.2, it was shown that performance criteria are continuous as the optimizer changes. To demonstrate this fact, the one-step optimizers for DE and PSO were convexly combined to generate a line in optimizer space, given by

$$\mathcal{G}_\alpha[t,f] = \alpha \mathcal{PSO}\!<\!-.5, 2, 2, 100\!>\![t,f] + (1-\alpha)\mathcal{DE}_{\text{rand}}\!<\!.2, .2, 100\!>\![t,f], \tag{8.5}$$



(a) $\phi_1(\text{NM-R}, f_\alpha)$, $\alpha \in (0, 1)$

(b) $\zeta_T(\text{NM-R}, f_\alpha)$, $\alpha \in (0, 1)$



(c) $\hat{\psi}^N_\epsilon(\text{NM-R}, f_\alpha)$, $\alpha \in (0, 1)$

(d) $\sigma^N_\epsilon(\text{NM-R}, f_\alpha)$, $\alpha \in (0, 1)$

Figure 8.8: Change in performance by NM-R as the objective changes smoothly from *langerman* ($\alpha = 0$) to *shekel* ($\alpha = 1$). The x-axis ranges over values of α, the y-axis over performance values. The first standard deviation is also plotted on either side of the performance where possible. Panels show the performance criteria ϕ_1, ζ_T, $\hat{\psi}^N_\epsilon$, and σ^N_ϵ, respectively. As predicted by the theory, performance on these optimizers changes smoothly and nonlinearly as a function of the objective.

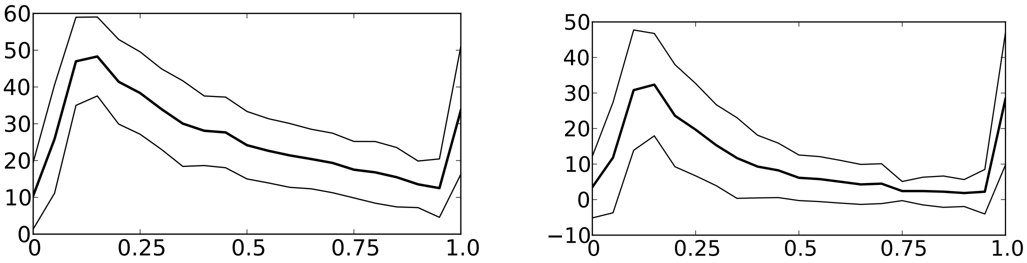
recalling $\mathcal{PSO}$ from Equation 4.37 and $\mathcal{DE}$ from Equation 4.43. The optimizer $\mathcal{G}_\alpha$ was tested with 200 trials on *schwefel* for $\alpha = 0.0, 0.05, 0.10, \ldots, 0.95, 1.0$. PSO outperforms DE on *schwefel* in general.

Figure 8.9 shows the performance of $\mathcal{G}_\alpha$ on *schwefel* for various values of α. Once again, the change in performance is smooth but non-linear, as predicted by the theory. The most interesting aspect is that although performance initially worsens for α in $(0, 0.15]$, it then improves consistently until $\alpha = 0.95$. In Figure 8.9(b), it can be seen that at $\alpha = 0.95$, $\mathcal{G}_\alpha$ outperforms both PSO and DE, although the result is statistically insignificant. Given that PSO is significantly better than DE on *schwefel* for most of the performance criteria, it is surprising that the best values of α are closer to DE rather than PSO. Convex combinations of optimizers were proposed as part of the formal analysis in Chapter 3, and the theory developed in Chapter 7 predicted convex combinations might outperform pure algorithms. The result of this experiment provides further evidence to support this claim. This discovery reinforces the value of the formal approach adopted in this dissertation.

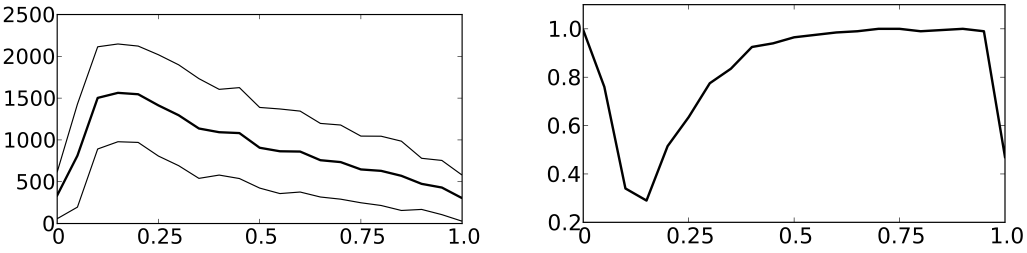
As mentioned, the fact that the best performance occurs for α other than zero or one confirms the conjecture in Section 7.2.1 that convex combinations of existing optimizers may outperform the optimizers being combined. Consequently, the problem of *convex control* of optimizers is worthy of further study. The problem of convex control may be stated as follows: given a bank of optimizers $\mathcal{G}_1, \ldots, \mathcal{G}_N$, an objective f, and a performance criterion ϕ, find the convex combination $\hat{\alpha} \in \mathbb{R}^N$ that minimizes $\phi(\mathcal{G}_\alpha, f)$, where $\mathcal{G}_\alpha = \sum_i \alpha_i \mathcal{G}_i$. This topic is discussed again briefly in Chapter 14.

8.4.3 An Example of Discontinuity

The proofs of continuity in Section 7.3.1 contained several conditions that must be met to guarantee continuity. It was stated that discontinuities can be expected at functions with substantial plateaus. There are two reasons that support this claim. First, functions with plateaus induce trajectories of ambiguous fitness that cause several specific optimizers to be discontinuous on sets of positive $\mathcal{G}_f$-measure. Second, functions with plateaus can introduce discontinuities in the hitting time even for optimizers that are continuous everywhere.



(a) $\phi_1(\mathcal{G}_\alpha, schwefel)$, $\alpha \in (0,1)$ (b) $\zeta_T(\mathcal{G}_\alpha, schwefel)$, $\alpha \in (0,1)$



(c) $\hat{\psi}_\epsilon^N(\mathcal{G}_\alpha, schwefel)$, $\alpha \in (0,1)$ (d) $\sigma_\epsilon^N(\mathcal{G}_\alpha, schwefel)$, $\alpha \in (0,1)$

Figure 8.9: Change in performance as the optimizer changes smoothly from PSO with $\omega = -.5, \phi_g = \phi_p = 2$ ($\alpha = 0$) to DE with CR=.2, F=.2 ($\alpha = 1$). The x-axis ranges over values of α, the y-axis over performance values. The first standard deviation is also plotted on either side of the performance where possible. The panels show the performance criteria ϕ_1, ζ_T, $\hat{\psi}_\epsilon^N$, and σ_ϵ^N, respectively, with $\epsilon = 25$ for *schwefel*. As predicted by the theory, performance on these optimizers changes smoothly and nonlinearly as a function of the optimizer. Interestingly, at $\alpha = .95$, $\mathcal{G}_\alpha$ outperforms PSO and DE on ζ_T, although the result is not statistically significant. Convex combinations of algorithms were formally proposed in this dissertation, and Section 7.2.1 suggested that convex combinations may outperform pure algorithms. This example validates this conjecture and confirms the value of the formal approach in this dissertation.

To demonstrate the discontinuities that occur as the objective passes through a continuous function, a new objective on $\mathbb{R}^d$ was created, *triangle*, defined on $(-30, 30)^d$ by

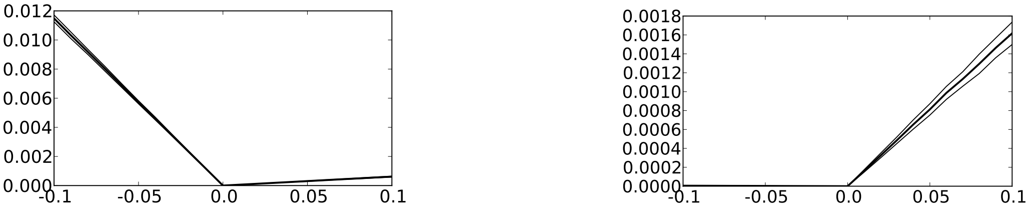
$$triangle(x) = \min_i \left[1 - \frac{1}{30}|x_i|\right]. \tag{8.6}$$

This objective function is a d-dimensional simplex with height 1 and base width 60. A range of objectives was then defined by $t_\alpha(x) = \alpha\, triangle(x)$ for $\alpha = -0.1, -0.09, -0.08, \ldots, 0.09, 0.1$. When $\alpha < 0$, the minimal values of $t_\alpha(x)$ occur at 0. When $\alpha > 0$, the minimal values of t_α are around the boundary of the space. When $\alpha = 0$, every point has minimal value. Thus as α passes through zero, the minimal points shift discontinuously.

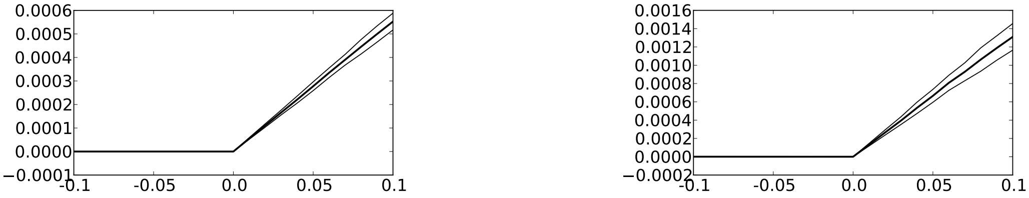
Figure 8.10 shows the values of the performance criteria ϕ_1, ζ, $\hat{\psi}_\epsilon^N$, and σ_ϵ^N as α runs from -0.1 to 0.1 for two optimizers in 25 dimensions. The two optimizers were (1) DE and (2) a real-coded genetic algorithms with proportional selection, uniform crossover, and gaussian mutation, named rGA-2 to distinguish it from rGA, which used ranking selection. DE is potentially discontinuous when $\alpha = 0$, but rGA-2 is continuous on all of $C[\mathbb{R}^{25}]$, which includes t_α for all values of α. The objective t_0 fails the requirements of Corollary 7.3.4, and so the performance of $\hat{\psi}_\epsilon^N$ and σ_ϵ^N each have a potential discontinuity at $\alpha = 0$, which is realized for DE in $\hat{\psi}_\epsilon^N$ and σ_ϵ^N for DE and is visible in Figures 8.10(f) and 8.10(h). For these graphs, ϵ was set at 0.0001.

The objective t_0 fails the requirements of Corollary 7.3.4, and so the performance of $\hat{\psi}_\epsilon^N$ and σ_ϵ^N each have a discontinuity at $\alpha = 0$ on DE (right panels) but not on rGA-2 (left panels). The plots are interpolated, but careful inspection of Figure 8.10(g) shows that σ_ϵ^N descends below 1 to the right of zero (it is still equal to 1 at $\alpha = 0.01$), whereas Figure 8.10(h) jumps discontinuously from 1 to 0 between $\alpha = 0$ and $\alpha = 0.01$.

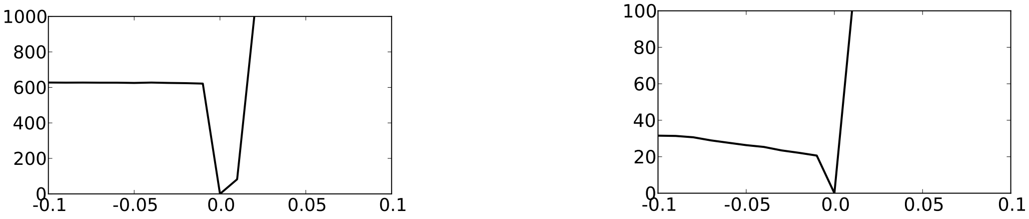
The hitting time for DE drops to zero as α approaches zero from the left, and the success probability is constant at one. From the right, the hitting time is fixed at $N = 250,000$, and the success probability is zero. The graphs for DE appear left continuous, but jump discontinuously to the right of zero. As noted above, the plots are interpolated. At $\alpha = 0.01$, DE immediately has $\hat{\psi}_\epsilon^N > 100$ versus values of $\hat{\psi}_\epsilon^N < 50$ on the left. In contrast, the value



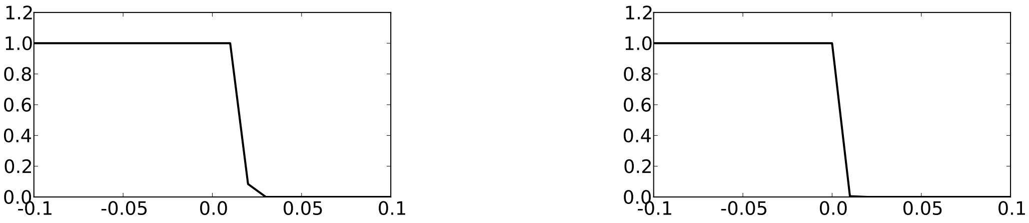
(a) $\phi_1(\text{rGA-2}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$ (b) $\phi_1(\text{DE}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$



(c) $\zeta_T(\text{rGA-2}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$ (d) $\zeta_T(\text{DE}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$



(e) $\hat{\psi}_\epsilon^N(\text{rGA-2}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$ (f) $\hat{\psi}_\epsilon^N(\text{DE}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$



(g) $\sigma_\epsilon^N(\text{rGA-2}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$ (h) $\sigma_\epsilon^N(\text{DE}, t_\alpha)$, $\alpha \in (-0.1, 0.1)$

Figure 8.10: Change in performance of rGA-2 and DE for zero-centered prisms of different heights in 25 dimensions, measured on t_α from Equation **??** for 21 values of α evenly spaced in the interval $[-0.1, 0.1]$. The x-axis ranges over values of α, the y-axis over performance values. Panels in the left column show the performance of rGA-2, and panels in the right column show the performance of DE. The rows show the performance criteria ϕ_1, ζ_T, $\hat{\psi}_\epsilon^N$, and σ_ϵ^N, respectively. DE has a discontinuity at t_0.

of $\hat{\psi}_\epsilon^N$ for rGA-2 also descends to zero from the left, but on the right it still has $\hat{\psi}_\epsilon^N$ relatively small at $\alpha = 0.01$. Notice that ϕ_1 and ζ are continuous for both optimizers despite the discontinuity in $\hat{\psi}_\epsilon^N$ and σ_ϵ^N due to Corollaries 7.3.2 and 7.3.3. Once again, theoretical inquiry correctly predicted potentially useful information about the performance of the optimizers.

8.5 Principal Components Analysis

Chapter 3 emphasized that the space of optimizers is a vector space with well-defined notions of distance between any two optimizers. In this section, a simple visualization of the space will be given for the optimizers that were included in the experiments. This visualization relies on the fact that there is a performance-based duality between optimizers and objective functions. This duality will be formally explored in Section 10.2. In essence, the results that will be presented in that section indicate that for a given performance criterion ϕ, the average performance against an objective function under ϕ forms a line through the vector space of long-running optimizers introduced in Chapter 6. The set of all possible distributions over objective functions induces an uncountable linear basis over optimizer space dependent on ϕ. The performance of an optimizer on a particular objective is thus a projection onto one of the components of this basis.

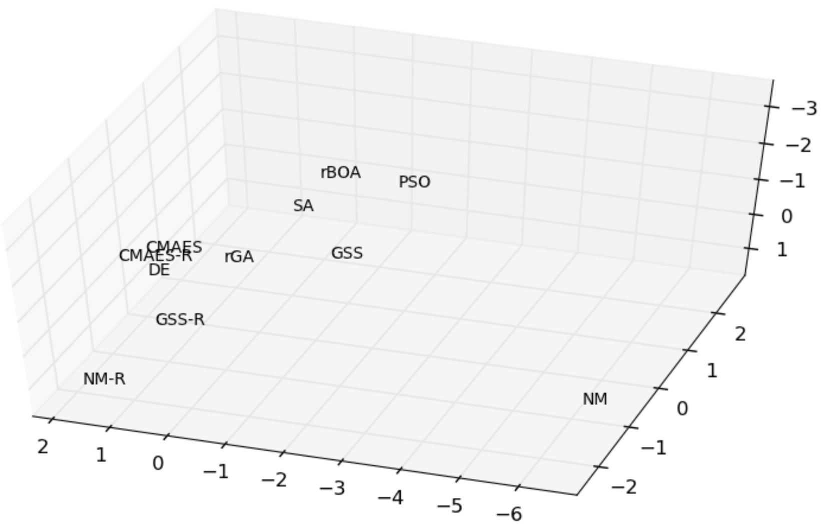
This property of optimizers was studied in the context of genetic algorithms by Ashlock [10]. They developed a set of test optimizers by varying the crossover rules of a genetic algorithm. The resulting optimizers were then run on an array of benchmarks. For a given crossover rule, the tuple $\left(\hat{\psi}_\epsilon^N, \sigma_\epsilon^N, \mathrm{Var}(\tau_\epsilon)\right)$ was recorded for each benchmark. The set of all tuples for each crossover rule was treated as a signature of the crossover rule unique to it. The system of Ashlock et al. works because of the duality described in the prior paragraph, but its effectiveness is not limited to genetic algorithms. It can also be applied to any performance criterion.

To demonstrate the results of such a characterization visually, the performance of each of the optimizers tested was used to create an array of performance values for each of the performance criteria ϕ_1, ζ_T, $\hat{\psi}_\epsilon$, and σ_ϵ^N. Principal components analysis (PCA) was used to project the performance into a three-dimensional space. PCA requires a square matrix, and there were thirteen algorithms and twelve benchmarks. To make a square matrix, the values for CG were excluded, so that the values for each performance criterion constituted an 12×12 matrix. For the performance criteria ϕ_1, the influence of CG-R on PCA was so strong that it skewed the visualization, and so PCA was performed without for this performance criteria, leaving an 11×11 matrix.

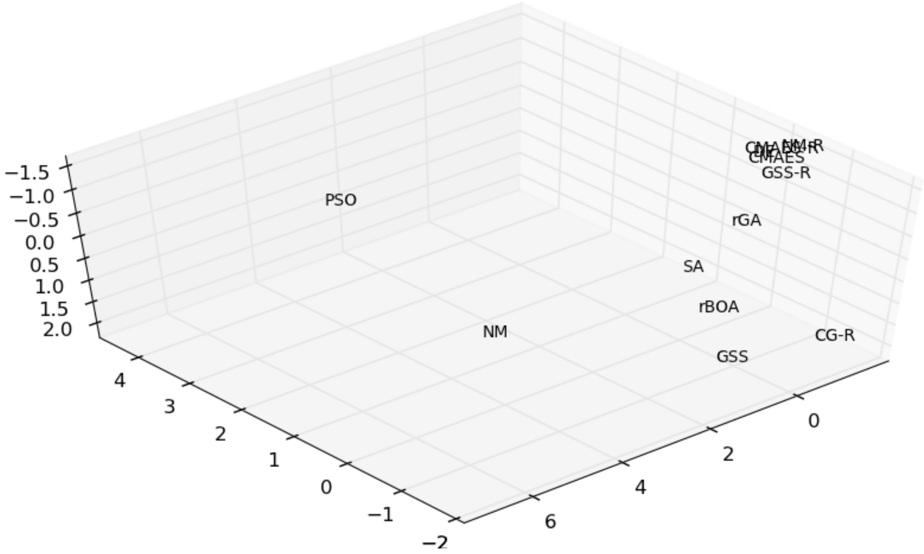
Principal components analysis (PCA) was applied to this matrix to create a 12-dimensional basis projection such that the earlier components have larger eigenvalues. For ϕ_1, the first three components found by PCA had an average range of 5.64. The remaining eight components had an average range of 1.39, so that the first three components do capture a substantial amount of the variation; results for other performance criteria were similar. These components were plotted in a three-dimensional scatterplot in Figures 8.11–8.13.

Reviewing the results for ϕ_1, the first PCA component separates NM ($x = -5$) from PSO ($x = -2.5$) and the rest ($x \in (0, 2)$). The second PCA component separates rBOA, GSS, SA, and rGA ($y < 0$) from the restarted optimizers, NM, DE, and CMA-ES ($y > 0$). The third component separates PSO ($z = -3$) from the rest ($z > -1$). More generally, for ϕ_1, (1) the restarted optimizers other than CG-R group together, (2) CMA-ES and DE are relatively close to each other and are closest to the restarted optimizers, and (3) SA and rGA generally appear together.

When CG-R is included for ϕ_1, then the first PCA component has a range of 10.75, as opposed to a range of 6.35 without it. Additionally, CG-R is located at -9.27 on this scale, whereas the other 11 algorithms fall between 0 and 1.45. Thus ϕ_1 strongly separates CG-R from the other algorithms, matching the intuition that gradient-based methods should behave in a noticeably different manner than gradient-free methods. If both CG and CG-R are included, with rBOA omitted, then a similar separation occurs, except that the first component separates CG and CG-R from the other methods, and the second component separates CG from CG-R. These distinctions are shown in Figure 8.12.



(a) First three PCA components of optimizers in the ϕ_1 basis, without CG-R



(b) First three PCA components of optimizers in the ζ_T basis

Figure 8.11: PCA plots for the test algorithms on the benchmarks. These plots show the proximity between various optimizers based on their performance on ϕ_1 and ζ_T. The resulting layout of optimizers reveals interesting new relationships among the algorithms.

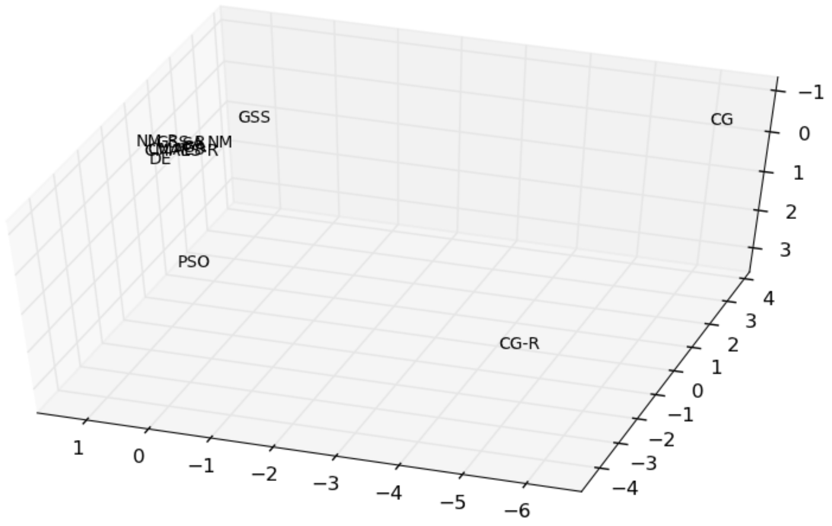


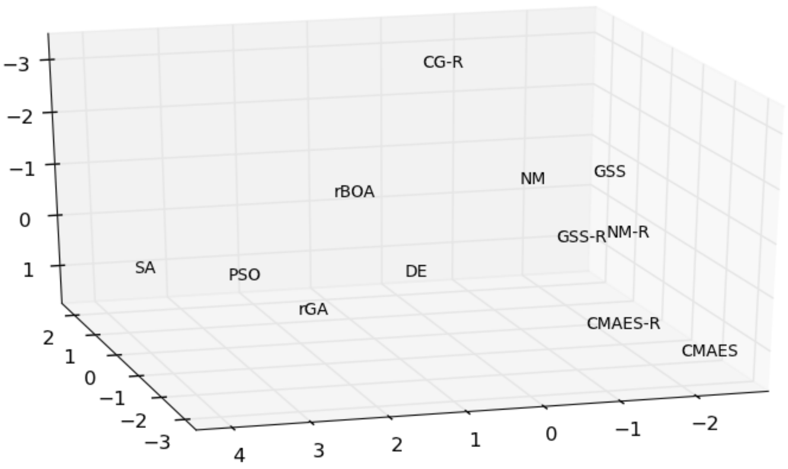
Figure 8.12: The first three PCA components in the ϕ_1 basis, with CG and CG-R included. The first component separates conjugate gradient descent from the other methods, and the second component separates the restarted version from the non-restarted version. This plot shows that CG and CG-R are indeed distinct from the other methods in terms of performance on ϕ_1.

The salient features for ζ_T resemble those for ϕ_1, without the disruptive influence of CG-R. The first component also isolates NM ($x = 6.5$) from the other methods ($x < 3$). The second component isolates PSO ($y = 4$), and the third component separates the restarted algorithms (except CG-R), CMA-ES, and DE ($z = -1.5$) from the others ($z > 0$). Once again, DE and CMA-ES are close both to each other and to the restarted optimizers. Also, SA and rGA are near to each other. CG-R does not cluster with the other restarted optimizers, in part because it performs worse than the others on ζ_T, particularly on problems like *log-ackley* and *weierstrass*.

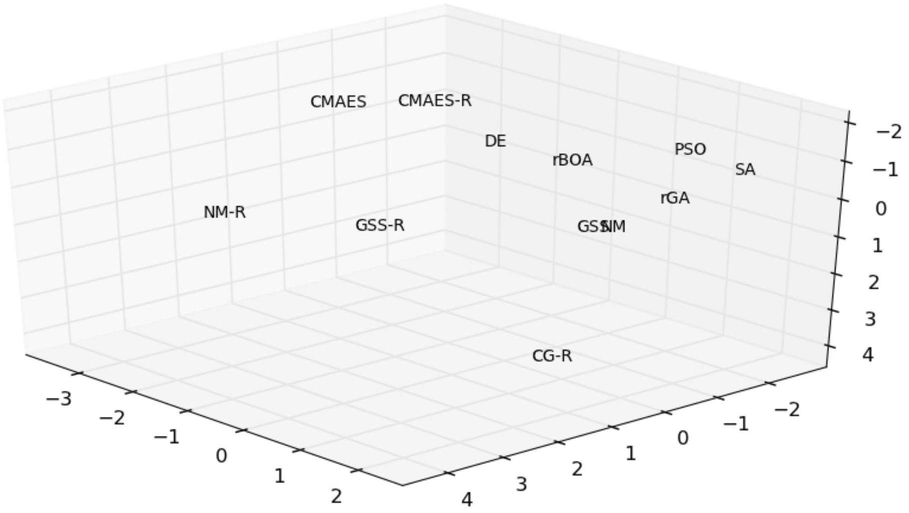
As might be expected, $\hat{\psi}^N_\epsilon$ separates optimizers first based on the convergence speed. NM, GSS, CMA-ES, and all the restarted optimizers have $x > 0$, while DE has $x = 0$ and the other algorithms have $x < 0$. The third component has CG-R at one end ($z = -3$) and quasi-evolutionary methods except rBOA at the other ($z = 1$), with direct search methods clustered together at the center ($z \in (-1, 0)$). Interestingly, this projection places the restarted version of optimizers close to the version without restarts in each case. Notably, SA and rGA are still relatively nearby for $\hat{\psi}^N_\epsilon$.

As for σ^N_ϵ, the first component separates the all of the restarted optimizers off from the rest. The second component places NM-R on the far negative side ($y = -3$) and SA on the other extreme ($y = 2$), with the rest distributed evenly. The third component separates CG-R ($z = 4$) from the others ($z < 2$). SA and rGA are still close together, and DE is as close to CMA-ES as it is to any other algorithm.

The graphs in Figures 8.11–8.13 thus demonstrate that the formal analysis of optimizers and their performance can enable new ways of looking at the relationships between optimizers. The picture that emerges provides insights that are not predicted by the origins of these optimizers. For example, simulated annealing and genetic algorithms are close in all of the graphs above. Even though such a result is unintuitive, it is substantiated by two theoretical observations: Theorem 4.2.3, which states that the (1+1)–ES is the norm-limit of simulated annealing, and the discussion of Expected Proportional Selection in Section 11.1.3. Thus theoretical inquiry is a useful tool for uncovering the connections between different optimizers.



(a) First three PCA components of optimizers in the $\hat{\psi}_\epsilon^N$ basis



(b) First three PCA components of optimizers in the σ_ϵ^N basis

Figure 8.13: PCA plots for eleven algorithms on eleven benchmarks. These plots show the proximity between various optimizers based on their performance on $\hat{\psi}_\epsilon^N$ and σ_ϵ^N. The resulting layout of optimizers reveals interesting new relationships among the algorithms.

8.6 Conclusion

The experiments in this chapter substantiated the theoretical analysis of performance undertaken in Chapter 7. These results demonstrate how theoretical analysis can suggest the existence new phenomena that can be observed experimentally. The formal approach adopted in this text makes it possible to compare algorithms using novel techniques that yield unforeseen insights, as when convex combinations were shown to outperform pure algorithms in some cases.

In earlier sections of this chapter, certain optimizers were shown to outperform others on the benchmarks. For example, the restarted algorithms, DE, and CMA-ES collectively perform much better than PSO, rGA, rBOA, SA, GSS, and NM. To the extent that such claims are restricted to the experiments performed, they cannot be disputed. But how will these optimizers perform on practical objectives on which they have not previously been tested? The next chapter extends the No Free Lunch theorems to infinite-dimensional search domains in order to answer this question: In any domain where learning is possible, there are always some optimizers that are better than others. What is observed experimentally in this chapter is thus proven theoretically in the next.

Chapter 9

No Free Lunch Does Not Prevent General Optimization

As was discussed in Chapter 2, a large number of heuristic optimization methods have been developed that attempt to locate the optimum of an arbitrary objective function automatically using only the sequence of objective values along an iterative path. Which of these optimizers is the best was a substantial focus of experimental and theoretical research for several decades. Then, in 1995, Wolpert and Macready published the first of the No Free Lunch (NFL) Theorems, proving that all non-repeating optimizers perform equivalently when averaged over all problems in a finite space [217]. Gradually, the assumptions and conclusions of NFL have been explored more thoroughly, and its overall impact has turned out to be much less destructive than was originally thought. In this chapter, the history is first reviewed, and then NFL is adapted to the formal setting introduced in the previous chapters. It is shown that NFL still applies in arbitrary measure spaces, and the exact conditions that lead to NFL are articulated and proven. These conditions generally make learning impossible, which is an absurd assumption for real-world problems. Thus general real-world problems cannot be subject to NFL.

9.1 Overview of No Free Lunch

The NFL theorems were first discovered by Wolpert and Macready in the context of search. Their 1995 paper concluded that all search heuristics pay for good performance on some datasets by performing poorly on other datasets, with average search quality over all datasets being constant in general [217]. In the same year, Radcliffe and Surrey applied Wolpert and Macready methods to obtain a similar result for optimization [162]. Two years later, Wolpert and Macready's published a proof that the average probability of obtaining a particular trajectory of objective values is independent of the optimizer selected. Since that time, a number of refinements and extensions have been produced [12, 48, 59, 60, 100, 171, 180, 181]. The relevant history is reviewed in this section.

9.1.1 NFL Basics

Wolpert and Macready treated an optimization algorithm as a deterministic function $a : \mathcal{T}[X \times Y] \to X$ on a finite search space X with objective values in a finite, strictly ordered set Y [218]. An algorithm a is non-repeating if $a(t) = x$ implies that $x \notin t$. Using their notation, they proved the following theorem:

Theorem 9.1.1 (No Free Lunch – Wolpert and Macready, 1997). *For any two iterative optimizers a_1 and a_2 that are non-repeating and all $m \leq |X|$,*

$$\sum_f \mathbb{P}(d^y_m \mid f, m, a_1) = \sum_f \mathbb{P}(d^y_m \mid f, m, a_2), \tag{9.1}$$

where $d^y_m \in \mathcal{T}[Y]$ is a sequence of objective values of length m, and f ranges over the function space Y^X.

The definition of an algorithm used by Wolpert and Macready is isomorphic to the set $\mathcal{DF} \cap \mathcal{O}_{\text{tr}}$ with $Y \subset \mathbb{R}$, where $\mathcal{DF}$ is the set of deterministic algorithms defined in Section 5.2 and $\mathcal{O}_{\text{tr}}$ is the set of trajectory-restricted optimizers from Chapter 3. The isomorphism is given explicitly by $\mathcal{A}[t, f](dx) = \delta_{a(t,f(t))}(x)$ where $f(t) \in \mathcal{T}[\mathbb{R}]$ is the trajectory formed by evaluations of t, i.e. $f(t)^i = f(t^i)$.

As Wolpert and Macready observed, it can easily be seen that the NFL Theorem applies to stochastic optimizers just as much as to deterministic ones by observing that X and Y are finite and taking a weighted sum over the possible algorithm outputs on either side of Equation 9.1 [218]. As a result, NFL applies to all of $\mathcal{O}_{\text{tr}}$ for finite search domains. The requirement that the set Y be finite and ordered means that Y can always be embedded into $\mathbb{R}$. Based on these observations, the NFL Theorem can be restated using the notation developed in Chapter 6.

Theorem 9.1.2 (No Free Lunch – Restated)**.** *Let $\mathcal{G}, \mathcal{G}' \in \mathcal{O}_{\mathrm{tr}}$ be non-repeating almost surely, and let X be finite. Let $Y \subseteq \mathbb{R}$ be finite as well, and let $\mathcal{F} = Y^X$ be the space of functions on X restricted to the finite set Y. Then for all $m \leq |X|$ and all $y \in \mathcal{T}[Y]$,*

$$\sum_{f \in \mathcal{F}} \mathcal{G}_f(\{x : f(x_1^m) = y\}) = \sum_{f \in \mathcal{F}} \mathcal{G}'_f(\{x : f(x_1^m) = y\}). \tag{9.2}$$

The plain meaning of this theorem is that no optimizer performs better than any other. The probability of attaining any level of performance on average is a constant independent of the algorithm used. As will be seen, NFL holds because the function space Y^X is incompressible. Observing the objective value at any point provides no information about the objective value at any other point. That is, NFL makes learning impossible.

The NFL Theorem above has three main limitations. First, it assumes that all functions are equally likely. Second, it only applies to finite spaces. Third, it only applies to algorithms that do not repeat points. The next few sections address the first two points in order. The third limitation will be relaxed in Section 9.3.

9.1.2 Function Priors and Closure Under Permutation

In order to provide the context for generalizing their result, Wolpert and Macready stated a more general equality,

$$\sum_f \mathbb{P}(d_m^y \mid f, m, a_1)\mathbb{P}(f) = \sum_f \mathbb{P}(d_m^y \mid f, m, a_2)\mathbb{P}(f), \tag{9.3}$$

in which a weighted sum replaces the average, with a prior distribution over functions $\mathbb{P}(f)$. If $\mathbb{P}(f) = 1/|Y^X|$, then Equation 9.1 is recovered. They conjectured at the time that the general equality would hold for many if not most function priors, particularly priors that place positive probability on a large number of functions. In retrospect, this claim was too expansive. The equality holds only for very few function priors that satisfy strict conditions.

The first result in this direction was obtained by Igel and Toussaint in 2004 [100]. Radcliffe and Surrey had previously proven an NFL theorem by using permutations on the search space [162]. A permutation $\pi : X \to X$ can be applied to an objective f to obtain a new objective $f \circ \pi$ that shuffles the inputs. A set of functions $\mathcal{F} \subseteq Y^X$ is closed under permutation (c.u.p.) if for any permutation π, $f \in \mathcal{F}$ implies that $f \circ \pi \in \mathcal{F}$. The uniform distribution over any c.u.p. set of functions is subject to NFL.

Theorem 9.1.3 (Sharpened NFL – Igel and Toussaint, 2004)**.** *Suppose $\mathcal{F} \subseteq Y^X$ is c.u.p. for X and Y finite. Then for any two optimizers $\mathcal{G}, \mathcal{G}' \in \mathcal{O}_{\mathrm{tr}}$ that are almost surely non-repeating, for all $m \leq |X|$ and all $y \in \mathcal{T}[Y]$,*

$$\sum_{f \in \mathcal{F}} \mathcal{G}_f(\{x : f(x_1^m) = y\}) = \sum_{f \in \mathcal{F}} \mathcal{G}'_f(\{x : f(x_1^m) = y\}). \tag{9.4}$$

Igel and Toussaint also asked how many c.u.p. subsets of Y^X exist. They proved that the percentage of subsets of Y^X that are c.u.p. is exactly

$$\frac{2^{\binom{|X| + |Y| - 1}{|Y|}} - 1}{2^{\left(|Y|^{|X|}\right)} - 1}. \tag{9.5}$$

This fraction vanishes double exponentially fast as $|X|$ and $|Y|$ increase. The obvious conclusion is that function priors subject to NFL are extraordinarily rare.

In some ways, the fact that NFL is rare does not make it less important. One of the main conclusions of NFL is that an optimizer's success on an optimization task depends on how well the optimizer is aligned with the class of problems likely to appear under $\mathbb{P}(f)$. Wolpert and Macready characterized this alignment with a loose statement of duality. The sums in Equation 9.4 may be regarded as the dot product of two vectors of size $|Y^X|$, one for $\mathbb{P}(d_m^y \mid f, m, a_1)$ and one for $\mathbb{P}(f)$. In this view, the performance of an optimizer is projected onto the function prior. An optimizer will perform better on function priors with which it is well-aligned. The paucity of NFL priors strengthens this interpretation by proving that opportunities for such alignment do occur. This perspective also holds in the infinite-dimensional setting and will be made rigorous in Section 10.2.

On the other hand, the sets of functions that are not c.u.p. can be quite large and general. The fact that an optimizer can be well-aligned with a very general set of functions weakens the claim that all optimizers perform equivalently, especially since closure under permutation seems to be an unreasonable assumption for any practical class of problems. This line of thought will pursued further below, where $\mathbb{P}(f)$ is allowed to be non-uniform and fully general.

9.1.3 Infinite Extensions of NFL

More recently, NFL has been extended to infinite spaces in different ways. Rowe et al. [171] used set-theoretic arguments based on permutations to show that an NFL property holds in spaces of arbitrary cardinality. Specifically, they show that every non-repeating optimizer has equivalent performance on any c.u.p. subset of functions in Y^X for X,Y of arbitrary cardinality. Their result generalizes NFL to infinite dimensions, but only in the case of uniform priors.

Auger and Teytaud extended NFL to countably and uncountably infinite domains using a measure-theoretic approach [12]. They introduced several variants and generalizations of NFL, including one based on a *random fitness function*. A random fitness function is defined as a random field over the search domain X, that is, a random variable that takes on values in $\mathbb{R}^X$. A random fitness F has the property $\mathcal{GNFL}$ if for any $m \leq |X|$ and any $\mathcal{G}, \mathcal{G}' \in \mathcal{O}_{\text{tr}}$ that are almost surely non-repeating, the two sets of random variables given by $\left(F(Z_1^{\mathcal{G}}), \ldots, F(Z_m^{\mathcal{G}})\right)$ and $\left(F(Z_1^{\mathcal{G}'}), \ldots, F(Z_m^{\mathcal{G}'})\right)$ are identically distributed. The meaning of these symbols will be made more rigorous in the next section, where $\mathcal{GNFL}$ corresponds to the *strong NFL* property.

With this definition, Auger and Teytaud proved that there exists a random fitness that possesses the $\mathcal{GNFL}$ property whenever the search domain is countably infinite. They also attempted to prove that there is no random fitness that has the $\mathcal{GNFL}$ property when $|X| = |\mathbb{R}|$ and concluded as a consequence that NFL does not apply to uncountable spaces. Unfortunately, their proof contains a fatal error, which it will be necessary to explain here. Auger and Teytaud correctly demonstrated that if the $\mathcal{GNFL}$ property holds for a random fitness F, then for any finite sequence $(x_1, \ldots, x_m) \subseteq X$, the random variables $F(x_1), \ldots, F(x_m)$ must be independent and identically distributed. This result holds and will be proven again in this chapter.

A theorem was presented stating that NFL cannot hold for any function prior when the search domain is $\mathbb{R}$, and that continuous lunches are therefore free. The claim was based on the assumption that NFL requires uncountable projections of a random fitness to be mutually independent, including projections onto uncountably many coordinates. Contradicting this assumption, NFL in fact only requires that the finite-dimensional projections of a random fitness be mutually independent, as is proven below. It will also be shown in in Theorem 10.4.1 that a random fitness subject to NFL always exists.

This mistake is easy to make. However, it is incorrect to consider uncountable collections of variables in this context, because the σ-algebra $\mathcal{B}[\mathbb{R}^X]$ is not rich enough to support such a conclusion. Even in the limit, cylinder sets can restrict at most countably many points. In an uncountable domain, there is a gap between any two points in a countable subset. Intuitively, this gap is big enough for a random function to forget where it came from, so that any countable collection of variables $\{F(x_i)\}_{i\in\mathbb{N}}$ can be independent, even if the random functions are continuous with probability one.

A rigorous approach to these issues follows in the next section.

9.2 NFL Preliminaries

The NFL theorems contain general statements about the average performance of all optimizers on all objective functions. To support such a broad claim, further theoretical structure and definitions are needed. In this section, the concept of a *random objective* and its *function prior* are defined. In addition, two variants of the NFL property are defined. *Strong NFL* indicates that the sequence of objective evaluations must be identically distributed under the function prior for any pair of optimizers. *Weak NFL* requires that all optimizers have the same average performance on some performance criterion. Finally, the properties of function priors are defined that can be used to state necessary and sufficient conditions for the NFL properties to hold.

9.2.1 Function Priors and Random Objectives

In the last section, the idea of a function prior $\mathbb{P}(f)$ was introduced as a probability measure ranging over objective functions. These function priors also correspond to the concept of a random test procedure described in Chapter 1. Suitable measures of this kind can be constructed by using the same technique that was used to construct the optimization process in Section 6.1.

Definition 9.2.1. *A function prior is a probability measure defined on the measurable space* $\left(\mathbb{R}^X, \mathcal{B}[\mathbb{R}^X]\right)$*, where* $\mathcal{B}[\mathbb{R}^X]$ *is the smallest* σ*-algebra containing the cylinder sets on* $\mathbb{R}^d$ *for arbitrary* d*.*

To define a function prior, it is sufficient to define a consistent family of finite-dimensional distributions, as in Definition 6.1.1 and the following text. Then, the Kolmogorov Extension Theorem guarantees the existence of the function prior exists as a measure on $\left(\mathbb{R}^X, \mathcal{B}[\mathbb{R}^X]\right)$ [50, 112, 113].

In order to support the duality result in Section 10.2, it is necessary to expand the concept of a function prior to include arbitrary finite signed measures on $\left(\mathbb{R}^X, \mathcal{B}[\mathbb{R}^X]\right)$; such a prior will be termed a *generalized function prior* when the distinction is important. The space of all generalized function priors will be denoted by $\mathcal{M}[\mathbb{R}^X]$ following the notation of Chapter 3. The space $\mathcal{M}[\mathbb{R}^X]$ is a Banach space under the norm $||\cdot||_{\mathcal{M}}$.

Consider the random variable $F(\omega, x)$ on $\omega \in \mathbb{R}^X$ defined by the coordinate mapping, i.e., $F(\omega, x) = \omega(x)$. Then F is a *random objective*, or equivalently, a *random fitness function.* The notation $F(x) = F(\omega, x)$ will refer to the random variable taking values on $\mathbb{R}$. The random variable $F^* = \inf_{x \in X} F(x)$ is the minimal value of F. The function prior corresponds to a distribution for F, and will be written as $\mathbb{P}_F$, so that $\mathbb{P}_F(A) = \mathbb{P}(F \in A)$ for any $A \in \mathcal{B}[\mathbb{R}^X]$. An expectation taken with respect to $\mathbb{P}_F$ is written $\mathbb{E}_{\mathbb{P}_F}[\cdot]$.

When considering theorems that pertain to optimization, it is not feasible to include arbitrary function priors. The minimum of a function prior may not be integrable, in which case the minimization task will not be defined for a set of functions with positive measure. Function priors will be termed *admissible* if $\mathbb{E}_{\mathbb{P}_F}[F^*]$ exists. For an admissible function prior, it also holds that

$\mathbb{P}_F(\{F^* > -\infty\}) = 1$. The set of admissible priors is closed under the vector operations of $\mathcal{M}[\mathbb{R}^X]$, and therefore the set of generalized admissible function priors is a vector subspace, denoted by $\mathcal{M}_a[\mathbb{R}^X]$. From this point forward, all function priors discussed in this text are assumed to be admissible.

The NFL properties can now be defined for function priors.

9.2.2 NFL Priors

As mentioned, the NFL theorems imply that under certain conditions, no optimizer outperforms any other optimizer when averaged uniformly over all fitness functions. The original theorems actually state that the trajectory of objective evaluations is independent of the choice of algorithm on average.

In this chapter, NFL will be defined with respect to the unique stopping sequence $(T_m)_{m\in\mathbb{N}}$ of Section 7.1.4. Recall that $T_m = T_m(z)$ is a stopping time indicating the index in an optimization sequence z at which m unique points have been evaluated. The uniquely stopped optimization process $(Z_{T_m})_{m\leq|X|}$ contains no repeated points if $T_m < \infty$. This stopped sequence will be used to replace the original NFL requirement that an optimizer should not repeat points. Thus the NFL results presented here apply in general to all algorithms that eventually propose new points by ignoring the repetitions. In addition, the NFL properties will be defined for arbitrary optimizer subsets $\mathcal{X} \subseteq \mathcal{PF}$, although they will only be proven initially for $\mathcal{O}_{\mathrm{tr}}$.

First, a strong criterion for NFL is stated with respect to the distribution of the evaluation process. Given an optimizer $\mathcal{G} \in \mathcal{PF}$, the *evaluation process* is the sequence of objective values given by $\left(F(Z^{\mathcal{G}}_{T_1}), \ldots, F(Z^{\mathcal{G}}_{T_m})\right)$. To satisfy the strong version NFL, the evaluation processes of all optimizers that eventually produce unique points must be identically distributed.

Definition 9.2.2. *A random objective F or its function prior $\mathbb{P}_F$ is strongly NFL on a set of optimizers $\mathcal{X} \subseteq \mathcal{PF}$ if for any $m \leq |X|$ and for any two optimizers $\mathcal{G}, \mathcal{G}' \in \mathcal{X}$ such that $T_m < \infty$ both $\mathcal{G}_f$-a.s. and $\mathcal{G}'_f$-a.s., $\left(F(Z^{\mathcal{G}}_{T_1}), \ldots, F(Z^{\mathcal{G}}_{T_m})\right)$ and $\left(F(Z^{\mathcal{G}'}_{T_1}), \ldots, F(Z^{\mathcal{G}'}_{T_m})\right)$ are identically distributed on $(\mathbb{R}^m, \mathcal{B}[\mathbb{R}^m])$.*

The strong NFL property is equivalent to the $\mathcal{GNFL}$ property of Auger and Teytaud [12]. It requires that the first m experimental objective values produced by the unique optimization trajectory of any two algorithms share the same distribution whenever the optimizers eventually produce at least m unique points.

At this stage, it is important to consider the nature of the distribution of $\left(F(Z^{\mathcal{G}}_{T_1}), \ldots, F(Z^{\mathcal{G}}_{T_m})\right)$, since this distribution must be handled formally in later proofs. This distribution is a joint distribution, since $F(Z_m)$ depends on the value of Z_m, and Z_m may depend on the value of F. For arbitrary optimizers in $\mathcal{PF}$, F must be sampled first, and Z may then be generated iteratively. In special cases, however, both F and Z may be sampled iteratively.

In the case of trajectory-restricted optimizers, $\mathcal{X} = \mathcal{O}_{\text{tr}}$, both Z and F may be sampled iteratively. First, Z_{T_1} is generated, then $F(Z_{T_1})$, then Z_{T_2} and $F(Z_{T_2})$, and so on. This procedure is possible because the optimizer depends only on the objective evaluations and nothing else. Sampling up to Z_{T_m} thus requires observing F at exactly m points. The distribution may be written down. As a first step, note that $T_1 = 1$, and so

$$\mathbb{P}\left(F\left(Z_{T_1}\right) \in dy_1\right) = \int_X \mathbb{P}_F(F(z_1) \in y_1)\mathcal{G}[\emptyset, 0](dz_1). \tag{9.6}$$

In this equation, $\mathcal{G}[\emptyset, 0]$ is used instead of $\mathcal{G}[\emptyset, F]$ to indicate that $\mathcal{G}$ is independent of F for the first step, since $\mathcal{G}$ is trajectory-restricted.

In order to expand the above result to T_m, it is necessary to integrate over the possible values of T_m. Recalling the set H_n of stopping trajectories for a stopping time from Equation 6.8 and using the fact that $T_m < \infty$ almost surely,

$$\mathbb{P}\left(\left(F\left(Z_{T_j}\right)\right)_{j=1}^m \in dy\right) = \sum_{n=1}^{\infty} \int_{H_n} \mathbb{P}_F\left(\left(F\left(t^{T_j(t)}\right)\right)_{j=1}^m \in dy\right) \prod_{i=1}^n \mathcal{G}[t_1^{i-1}, y_1^{i-1}](dt^i) \tag{9.7}$$

The term $\mathcal{G}[t_1^{i-1}, y_1^{i-1}]$ was used instead of $\mathcal{G}[t_1^{i-1}, F]$ to indicate that $\mathcal{G}$ depends only on the initial values of y, since it is trajectory-restricted. The distribution in Equation 9.7 will be used several times below. If a joint expectation is taken with respect to both $\mathbb{P}_F$ and $\mathcal{G}_F$, then the integrand should appear inside the integral of H_n, as in Proposition 9.2.1 below.

The strong NFL property can be weakened by requiring only that the average values of a particular performance criterion be equal. This property will be termed *weak NFL* because it corresponds to the equality under just one performance criterion.

Definition 9.2.3. *Given a performance criterion ϕ that is uniquely dependent on the error sequence, a random objective F or its function prior $\mathbb{P}_F$ is weakly NFL in ϕ on a set of optimizers $\mathcal{X} \subseteq \mathcal{PF}$ if for any two optimizers $\mathcal{G}, \mathcal{G}' \in \mathcal{X}$ such that $T_m < \infty$ both $\mathcal{G}_f$-a.s. and $\mathcal{G}'_f$-a.s. for all $m \leq |X|$ on which ϕ depends, it holds that $\mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}, F)] = \mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}', F)]$.*

The phrase "on which ϕ depends" in the definition of weak NFL accounts for performance criteria that are finitely uniquely dependent on the error sequence, in which case $\mathcal{G}$ and $\mathcal{G}'$ need only have $T_m < \infty$ at the values of m for which ϕ depends on $E^f_{T_m}$.

If a prior is strongly NFL, then it admits no statistical difference in the trajectories produced by running a pair of optimizers on a random objective. If the prior is only weakly NFL, there may be differences in the trajectories, but these differences disappear when integrating to measure performance on a particular performance criterion. For trivial performance criteria, such as $\phi(\mathcal{G}, f) = c$ with c constant, every prior is weakly NFL. In general, such performance criteria are uninteresting and uninformative. In practical terms, if a prior is weakly NFL on a large enough set of performance criteria, the implications do not differ greatly between the two. Importantly, strong NFL implies weak NFL.

Theorem 9.2.1. *Every strongly NFL prior is also weakly NFL on $\mathcal{O}_{\mathrm{tr}}$ over all uniquely dependent performance criteria.*

Proof. Suppose F is a strongly NFL random objective and ϕ is a performance criterion that is uniquely dependent on the error sequence. Suppose initially that ϕ is finitely uniquely dependent on the error sequence up to the M^{th} unique point. Let $\mathcal{G}$, $\mathcal{G}' \in \mathcal{O}_{\mathrm{tr}}$ such that $T_M < \infty$ both $\mathcal{G}_f$-*a.s.* and $\mathcal{G}'_f$-*a.s.* for some M on which ϕ depends.

Because ϕ is uniquely dependent,

$$\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[\tilde{h}\left(\left(E^f_{T_n}\right)_{n=1}^{M}\right)\right]$$

for some $\tilde{h}$ and the error process $E^f_{T_n}$. Also, since $\mathcal{G}, \mathcal{G}' \in \mathcal{O}_{\text{tr}}$, both $\mathcal{G}_f$ and $\mathcal{G}'_f$ can only depend on the objective value of the first M unique points by time T_M. Therefore, the expectation $\mathbb{E}_{\mathbb{P}_F}$ can be computed by integrating only over these m points plus the optimum, and with H_n as the stopping set for T_M, we have

$$\begin{aligned} \mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}, f)] &= \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}f}\left[\tilde{h}\left(\left(E^f_{T_n}\right)_{n=1}^{M}\right)\right] \\ &= \int_{\mathbb{R}^{m+1}} \tilde{h}\left((y_j - y^*)_{j=1}^{M}\right) \sum_{n=1}^{\infty} \int_{H_n} \prod_{i=1}^{n} \mathcal{G}[t_1^{i-1}, y_1^{i-1}](dt^i) \\ &\quad \times \quad \mathbb{P}_F\left(\left(F\left(t^{T_j(t)}\right)\right)_{j=1}^{M} \in dy, F^* \in dy^*\right) \end{aligned} \tag{9.8}$$

In these equations, the integrand $\tilde{h}$ was extracted from the inner integrals because it does not depend on t. Recognizing that the outer integral is taken with respect to the distribution of $(F(Z_{T_1}), \ldots, F(Z_{T_m}))$, which is shared between $\mathcal{G}$ and $\mathcal{G}'$, it follows that

$$\mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}, f)] = \mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}', f)]. \tag{9.9}$$

Therefore F is weakly NFL in ϕ.

To remove the assumption that ϕ is finitely uniquely dependent, recall that any uniquely dependent ϕ is progressively decomposable by Theorem 7.2.5. Therefore we may construct a sequence of performance criteria,

$$\phi_m(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[\sum_{n=1}^{T_m} h_m(Z_1^m, f)\right]. \tag{9.10}$$

At $m = \infty$, the sum comes out of the expectation, and so $\phi_m \to \phi$. Since ϕ is uniquely dependent on the error sequence, ϕ_m is finitely uniquely dependent on the error sequence. Consequently, $\mathbb{E}_{\mathbb{P}_F}[\phi_m(\mathcal{G}, f)] = \mathbb{E}_{\mathbb{P}_F}[\phi_m(\mathcal{G}', f)]$ for all suitable $\mathcal{G}, \mathcal{G}'$, and

$$\mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}, f)] = \lim_m \mathbb{E}_{\mathbb{P}_F}[\phi_m(\mathcal{G}, f)] = \lim_m \mathbb{E}_{\mathbb{P}_F}[\phi_m(\mathcal{G}', f)] = \mathbb{E}_{\mathbb{P}_F}[\phi(\mathcal{G}', f)]. \tag{9.11}$$

□

In this dissertation, the emphasis will be placed on strongly NFL priors, with secondary discussion of weakly NFL priors. The next section presents properties of function priors that are necessary and sufficient to draw conclusions about the strong and weak NFL properties.

9.2.3 Properties of Function Priors

As shall be demonstrated, a prior is weakly NFL on ϕ_w and ζ_{T_m} if it has uncorrelated paths and a constant mean at every evaluation point. A particular function prior is strongly NFL if it is identically distributed over the search space and if it is independent along all possible evaluation paths. An NFL prior implies that information from one evaluation at one point provides no information about evaluation at any other point. To illustrate, consider the Fibonacci sequence

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \dots$$

Under an NFL prior, there is no basis for claiming that the next number in this sequence is 233; it could just as likely be 234, 11, or $3,486,269,203$. NFL assumes a sequence is not correlated with itself.

Definition 9.2.4. *A random objective F or its prior $\mathbb{P}_F$ is path independent if for any finite collection of distinct points $x_1, \dots, x_n \subseteq X$, the collection $(F(x_i))_{i=1}^n$ is mutually independent. If F is not path independent, then it is path dependent.*

The name of path independence captures the intuition that no trajectory through the search space provides more information about the objective value of another point than any other. The objective values are independent of any such path. However, path independence is not sufficient to account for NFL priors. To be strongly NFL, a random objective must be identically distributed at each point. If it were not, then two different optimizers could result in distinct evaluation processes by exploiting the variations in objective values between two points in the search space. Path independence and identical distributions are the two main properties that hold for a strongly NFL prior. These two properties are in fact provably equivalent to strong NFL.

To be weakly NFL, a random objective may only need to satisfy a less stringent requirement. As an example, for the performance criteria ζ_T and ϕ_T, the random objective only needs to have a constant mean across all inputs and uncorrelated paths. Otherwise, one optimizer could outperform another on these criteria by prioritizing more optimal inputs.

Definition 9.2.5. *A random objective F or its function prior $\mathbb{P}_F$ is mean-constant if $\mathbb{E}_{\mathbb{P}_F}[F(x)] = \mathbb{E}_{\mathbb{P}_F}[F(y)]$ for all x, y in X.*

Notice that having a constant mean at each point and being identically distributed at each point are substantially different requirements. A function prior can be mean-constant and still have substantially different probability distributions at each point. For additively decomposable performance criteria, however, these differences may be integrated out to obtain the weak NFL property for ϕ_w and ζ_{T_m}. On the other hand, every identically distributed prior is also mean-constant. Thus any result that applies to mean-constant priors also applies to priors that are identically distributed.

For similar reasons, weak NFL holds for ϕ_w and ζ_{T_m} under a slightly different condition on the evaluation paths. These paths do not need to be independent for weak NFL; it is sufficient if they are uncorrelated.

Definition 9.2.6. *A random objective F or its prior $\mathbb{P}_F$ is path uncorrelated if for any finite collection of distinct points $x_1, \ldots, x_n \subseteq X$,*

$$\mathbb{E}_{\mathbb{P}_F}[F(x_1) \mid F(x_2), \ldots, F(x_n)] = \mathbb{E}_{\mathbb{P}_F}[F(x)].$$

One final edge case must be handled before moving on. Universally constant priors trivially possess the NFL property. However, some non-constant priors may have a *universal minimum.*

Definition 9.2.7. *A point $x \in X$ is a universal minimum of a function prior $\mathbb{P}_F$ if there exists an $x \in X$ such that the set $\mathcal{F}_x = \{f : f(x) = f^*\}$ has $\mathbb{P}_F$-probability one.*

If a prior has a universal minimum, then an optimizer can obtain perfect performance by guessing the minimum. This ability would violate NFL unless every point in X is a universal minimum, in which case every optimizer obtains perfect performance. Such a prior is termed *universally constant.* Mean-constant priors have a universal minimum if and only if they are universally constant.

Definition 9.2.8. *A function prior* $\mathbb{P}_F$ *is said to be universally constant if every* $x \in X$ *is a universal minimum.*

These two concepts coincide for mean-constant priors and, by extension, priors that are identically distributed at each point.

Lemma 9.2.2. *If a mean-constant prior* $\mathbb{P}_F$ *has a universal minimum then* $\mathbb{P}_F$ *is universally constant.*

Proof. Suppose $\mathbb{P}_F$ has a universal minimum at $x \in X$; then $\mathcal{F}_x = \{f : f(x) = f^*\}$ has $\mathbb{P}_F$-probability one. Then because $\mathbb{P}_F$ is mean-constant, $\mathcal{F}_y = \{f : f(y) = f^*\}$ has $\mathbb{P}_F$-probability one for all $y \in X$. That is, $\mathbb{P}_F$ is universally constant. $\square$

Using these definition, a prior is strongly NFL if and only if it is identically distributed and path independent. A prior is weakly NFL over additively decomposable performance criteria if and only if it is mean-constant and path independent. Establishing these facts is the topic of the next section.

9.3 NFL Theorems

This section presents a series of results that culminate in the NFL Identification Theorems 9.3.7 and 9.3.8. The core ideas are built up in several lemmas and theorems, which are then aggregated into the primary results characterizing strong and weak NFL.

9.3.1 Implications of Strong and Weak NFL

The following two lemmas establish that strong NFL priors are identically distributed, and weak NFL priors over additively decomposable performance criteria are mean-constant.

Lemma 9.3.1. *Every strongly NFL prior on* $\mathcal{O}_{\mathrm{tr}}$ *is identically distributed at every point in* X.

Proof. Suppose that $\mathbb{P}_F$ is strongly NFL but not identically distributed. Then there exist $x, y \in X$ and $\mathbb{A} \in \mathcal{B}[\mathbb{R}]$ such that

$$\mathbb{P}(F(x) \in A) \neq \mathbb{P}(F(y) \in A). \tag{9.12}$$

Let $\mathcal{G}_x, \mathcal{G}_y \in \mathcal{O}_{\mathrm{tr}}$ with $\mathcal{G}_x[\emptyset, f](\{z\}) = \delta_x(z)$ and $\mathcal{G}_y[\emptyset, f](\{z\}) = \delta_y(z)$. Then it is immediate that

$$\mathbb{P}(F(Z_{T_1}^{\mathcal{G}_x} \in A) = \mathbb{P}(F(x) \in A) \neq \mathbb{P}(F(y) \in A) = \mathbb{P}(F(Z_{T_1}^{\mathcal{G}_y} \in A), \tag{9.13}$$

contradicting the assumption that $\mathbb{P}_F$ is strongly NFL. Therefore $\mathbb{P}_F$ is identically distributed at every point. $\square$

Lemma 9.3.2. *Every function prior that is weakly NFL on* $\mathcal{O}_{\mathrm{tr}}$ *over a non-trivial additively decomposable performance criterion is also mean-constant.*

Proof. Suppose ϕ is an additively decomposable performance criterion that is also uniquely dependent on the error sequence. Then ϕ depends on ζ_{T_m} for one or more values of m, since it is non-trivial. Thus if the results holds for ζ_{T_m} for arbitrary m, then it holds for all ϕ meeting the assumptions.

Using the same technique as in Lemma 9.3.1, if $\mathbb{P}_F$ is weakly NFL but not mean-constant, then there exist $x, y \in X$ such that

$$\mathbb{E}_{\mathbb{P}_F}\left[F(x) - F^*\right] > \mathbb{E}_{\mathbb{P}_F}\left[F(y) - F^*\right]. \tag{9.14}$$

Let $\mathcal{G}_x^m, \mathcal{G}_y^m \in \mathcal{O}_{\mathrm{tr}}$ such that $\mathcal{G}_x[t, f](\{z\}) = \delta_x(z)$ and $\mathcal{G}_y[t, f](\{z\}) = \delta_y(z)$ for any t containing exactly $m - 1$ unique points. Then

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_x^m, F\right)\right] = \mathbb{E}_{\mathbb{P}_F}\left[F(x) - F^*\right] > \mathbb{E}_{\mathbb{P}_F}\left[F(y) - F^*\right] = \mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_y^m, F\right)\right]. \tag{9.15}$$

The inequality in Equation 9.15 contradicts the assumption that $\mathbb{P}_F$ is weakly NFL in ϕ. So $\mathbb{P}_F$ is mean-constant. $\square$

Note that in Lemma 9.3.1 and Lemma 9.3.2, the proof did not depend strongly on the trajectory-restrictedness of the optimizer; it only required the existence of $\mathcal{G}_x$ and $\mathcal{G}_y$ meeting the description. The exact same proof holds for all of $\mathcal{O}_{\text{ir}}$ and for many subsets of $\mathcal{PF}$ that are of interest. These facts are explored with slightly more detail in Chapter 14, where the NFL Identification Theorem is extended to $\mathcal{O}_{\text{ir}}$. The next theorem, however, does depend heavily on trajectory restrictions.

9.3.2 NFL Implications on the Evaluation Path

This section shows that NFL implies that past evaluations can reveal no information about the outcome of future evaluations. This fact is one of the key points of this dissertation, since it is equivalent to the claim that NFL makes learning impossible. First, it is shown that strongly NFL priors are path independent.

Lemma 9.3.3. *Suppose $\mathbb{P}_F$ is a strongly NFL prior. Then $\mathbb{P}_F$ is path independent.*

Proof. The conditional probability of $F(Z^{\mathcal{A}}_{T_m})$ given $F(Z^{\mathcal{A}}_{T_1}),\dots,F(Z^{\mathcal{A}}_{T_{m-1}})$ is

$$\mathbb{P}\left(F(Z^{\mathcal{A}}_{T_m}) \in A \mid F(Z^{\mathcal{A}}_{T_1}),\dots,F(Z^{\mathcal{A}}_{T_{m-1}}) \in B\right) \tag{9.16}$$

for $A \in \mathcal{B}[\mathbb{R}]$ and $B \in \mathcal{B}[\mathbb{R}^{m-1}]$. The NFL property implies that the distribution of $F(Z^{\mathcal{A}}_{T_1}),\dots,F(Z^{\mathcal{A}}_{T_{m-1}})$ is independent of Z, and so Equation 9.16 may be written as a function $g^m_A(B)$ independent of $Z_{T_1},\dots,Z_{T_m}$.

Notice that $g^m_A(X)$ is the marginal distribution of $F(Z^{\mathcal{A}}_{T_m})$. The claim that $\mathbb{P}_F$ is path independent is equivalent to the claim that $g^m_A(B) = g^m_A(\mathbb{R})$ for all $m \leq |X|$, $A \in \mathcal{B}[\mathbb{R}]$, and all nonempty $B \in \mathcal{B}[\mathbb{R}^{m-1}]$. If $\mathbb{P}_F$ is universally constant, it is trivial that $g^m_A(B) = g^m_A(\mathbb{R})$. Thus we may assume that $\mathbb{P}_F$ is not universally constant.

Suppose that for some $m \leq |X|$, there exists $A \in \mathcal{B}[\mathbb{R}]$ and a nonempty $B \in \mathcal{B}[\mathbb{R}^{m-1}]$ such that $g^m_A(B) \neq g^m_A(\mathbb{R})$. Let C be the complement of B in $\mathbb{R}$, i.e. $C = \mathbb{R} \setminus B$, and then it follows that $g^m_A(B) \neq g^m_A(C)$, which is possible since $\mathbb{P}_F$ is not universally constant. The only remaining step is to exhibit two stochastic optimizers, one of which passes through B while the other traverses C. If such optimizers exist, then their evaluation processes are not identically distributed, contradicting the assumption that $\mathbb{P}_F$ was of class NFL.

The sets B and C are both nonempty, and so there are trajectories $t, t' \in X^m$ such that

$$F(t_1), \ldots, F(t_{m-1}) \in B \tag{9.17}$$
$$F(t'_1), \ldots, F(t'_{m-1}) \in C. \tag{9.18}$$

Let $\mathcal{A}$ be a deterministic optimizer that produces t with probability one regardless of the objective, and let $\mathcal{A}'$ produce t' in the same way. Then

$$\mathbb{P}\left(\left(F(Z^{\mathcal{A}}_{T_j})\right)_{j=1}^{m} \in A \times \mathbb{R}^{m-1}\right) = g^m_A(B) \tag{9.19}$$
$$\mathbb{P}\left(\left(F(Z^{\mathcal{A}'}_{T_j})\right)_{j=1}^{m} \in A \times \mathbb{R}^{m-1}\right) = g^m_A(C), \tag{9.20}$$

which contradicts the assumption that $\mathbb{P}_F$ was of class NFL since $g^m_A(B) \neq g^m_A(C)$. Therefore $\mathbb{P}_F$ is path independent. □

Next, any function prior that is weakly NFL in ζ_{T_m} is shown to be path uncorrelated, provided that it has no universal minimum. The proof of this statement is quite demanding, but it leads to one of the most important conclusions in this dissertation.

Theorem 9.3.4. *Suppose $\mathbb{P}_F$ is a function prior that is weakly NFL in the performance criterion ζ_{T_m} on $\mathcal{O}_{\text{tr}}$ for all $m < |X|$. Then if $\mathbb{P}_F$ has no universal minimum, it is path uncorrelated.*

Proof. For a trajectory $t \in \mathcal{T}[X]$, an evaluation trajectory $y \in \mathcal{T}[\mathbb{R}]$ with $|y| = |t|$, and a point $x \in X$, define

$$\hat{w}(x,t,y) = \mathbb{E}_{\mathbb{P}_F}\left[F(x) - F^* \mid F\left(t^1\right) = y^1, \ldots, F\left(t^{|t|}\right) = y^{|t|}\right] \tag{9.21}$$
$$\hat{u}(x,t,y) = \mathbb{E}_{\mathbb{P}_F}\left[F(x) - F^*\right] \tag{9.22}$$

so that $\hat{w}(x,t,y)$ is the average evaluation of x on $\mathbb{P}_F$ conditioned on t, y, and $\hat{u}(x,t,y) = \hat{u}(x)$ is the average evaluation of x without conditioning. If $\hat{w}(x,t,y) = \hat{u}(x,t,y)$ for all inputs, then $\mathbb{P}_F$ is path uncorrelated.

This fact will be demonstrated by showing that it is true over all countable sets in $\mathcal{B}_\tau$. Let $A \in \mathcal{B}_\tau$ be countably infinite, or choose A so that $|A| = |X|$ if X is finite. Note that any countable subset of X is $\mathcal{B}_\tau$-measurable because τ is Hausdorff. Next, it would be desirable to construct a measure λ over A so that $\hat{w}$ and $\hat{u}$ are λ-integrable. Unfortunately, it is not possible to do so, since $\hat{w}$ and $\hat{u}$ may be infinite on A. To work around this, define for $N = 1, 2, \ldots$

$$\hat{w}_N(x,t,y) = \hat{w}(x,t,y) \wedge N \tag{9.23}$$

$$\hat{u}_N(x,t,y) = \hat{u}(x,t,y) \wedge N \tag{9.24}$$

If $\hat{w}_N = \hat{u}_N$, then $\hat{w} = \lim_N \hat{w}_N = \lim_N \hat{u}_N = \hat{u}$. Thus it suffices to prove $\hat{w}_N = \hat{u}_N$ for arbitrary N. From this point on, let N be arbitrary and fixed.

Note that $\hat{w}_N, \hat{u}_N > 0$ because $\mathbb{P}_F$ has no universal minimum. Choose an enumeration $(a_n)_{i=1}^{\infty}$ of A. Define λ by

$$\lambda(X \setminus A) = 0 \tag{9.25}$$

$$\lambda(\{a_n\}) = 2^{-n} \quad \text{for all } n \tag{9.26}$$

so that every point in A has positive λ-probability. Then $\hat{u}_N, \hat{w}_N$ are λ-integrable since

$$\int_X \hat{u}_N(x)\, \lambda(dx) = \sum_n 2^{-n}\ \hat{u}_N(a_n) \ \leq\ N,$$

and similarly for $\hat{w}_N$.

These functions can be normalized into probability distributions over A, since they are λ-integrable:

$$w_N(x,t,y) = \hat{w}_N(x,t,y) \times \left(\int_{X\setminus t} \hat{w}_N(z,t,y)\, \lambda(dz)\right)^{-1} \tag{9.27}$$

$$u_N(x,t,y) = \hat{u}_N(x,t,y) \times \left(\int_{X\setminus t} \hat{u}_N(z,t,y)\, \lambda(dz)\right)^{-1}. \tag{9.28}$$

The integrals are taken of the set with t removed in order to avoid repetition in t in the next step. Define trajectory restricted optimizers $\mathcal{G}_1$ and $\mathcal{G}_2$ as follows:

$$\mathcal{G}_1[t,f](dx) = \begin{cases} w_N(x,t,y)\, \lambda(dx) & \text{if } x \notin t \\ 0 & \text{otherwise} \end{cases} \tag{9.29}$$

$$\mathcal{G}_2[t,f](dx) = \begin{cases} u_N(x,t,y)\, \lambda(dx) & \text{if } x \notin t \\ 0 & \text{otherwise.} \end{cases} \tag{9.30}$$

Then $\mathcal{G}_1$ leverages the conditional probabilities, whereas $\mathcal{G}_2$ does not. Both optimizers are defined so as to not repeat points. Then $T_m = m$, so

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_i, F\right)\right] = \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}_i F}\left[F\left(Z_m^*\right) - F^*\right] \tag{9.31}$$

for $i = 1, 2$ using the performance criterion ζ_{T_m}. Since $\mathbb{P}_F$ is weakly NFL,

$$\mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}_1 F}\left[F\left(Z_m^*\right) - F^*\right] = \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}_2 F}\left[F\left(Z_m^*\right) - F^*\right] \tag{9.32}$$

for all $m > 0$. Since $w(x, \emptyset, \emptyset) = u(x, \emptyset, \emptyset)$, this equation holds for $m = 1$. Now let $\mathcal{T}_m = \{t \in \mathcal{T} : |t| = m \,\text{and}\, t \,\text{contains no repetitions}\}$. Let t_1^i be the trajectory formed by taking the first i elements of t, and let $f(t)$ be the trajectory in $\mathcal{T}[\mathbb{R}]$ formed by evaluating f over t in order. Further, let $f(t^*)$ be the minimum value of f over a trajectory t. Then

$$\begin{aligned} \mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_2, F\right)\right] &= \int_{\mathbb{R}^X}\int_{\mathcal{T}_m}[f(t^*) - f^*]\left[\prod_{i=1}^m u_N\left(t^i, t_1^{i-1}, f(t_1^{i-1})\right)\lambda\left(dt^i\right)\right]\mathbb{P}_F(df) \\ &= \int_{\mathcal{T}_m}\int_{\mathbb{R}^X}[f(t^*) - f^*]\,\mathbb{P}_F(df)\left[\prod_{i=1}^m u_N(t^i)\lambda\left(dt^i\right)\right], \end{aligned} \tag{9.33}$$

where the integrands can be reversed because u_N is independent of f and all terms are positive. Define $k_m(t, f)$ on $\mathcal{T}_m$ by

$$k_m(t, f) = [f(t^*) - f^*]\prod_{i=1}^{m-1} u_N\left(t^i\right).$$

Next define a measure over $\mathcal{T}_m \times \mathbb{R}^X$,

$$\kappa_m(dt, df) = k_m(t, f)\,\mathbb{P}_F(df)\prod_{i=1}^m \lambda\left(dt^i\right),$$

so that $\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_2, F\right)\right]$ is the norm of $\tilde{u}_N(t) = u_N(t^{-1})$ in $L^1\left[\mathcal{T}_m \times \mathbb{R}^X, \kappa_m\right]$,

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_2, F\right)\right] = \int_{\mathcal{T}_m}\int_{\mathbb{R}^X}\tilde{u}_N(t, f)\,\kappa_m(dt, df) = ||\tilde{u}_N||_{\kappa_m}.$$

Fix $m = 2$. By a similar sequence of equations using $\tilde{w}_N(t, f) = w_N\left(t^i, t_1^{i-1}, f(t_1^{i-1})\right)$,

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_2}\left(\mathcal{G}_1, F\right)\right] \;=\; \int_{\mathcal{T}_2}\int_{\mathbb{R}^X}\tilde{w}_N(t, f)\,\kappa_2(dt, df) \;=\; ||\tilde{w}_N||_{\kappa_2},$$

and therefore since $\mathbb{P}_F$ is weakly NFL, $w_N(x,t,y) = u_N(x)$ almost everywhere (*a.e.*) in κ_2 if $|t| = 2$. This result can be extended to all $m > 1$ by induction. For the induction hypothesis, suppose that $w_N(x,t,y) = u_N(x)\, a.e.$ in κ_{m-1}. Then u_N can be substituted for w_N on trajectories shorter than m. Therefore

$$[f(t^*) - f^*] \prod_{i=1}^{m-1} w_N\left(t^i, t_1^i, f(t_1^{i-1})\right) = [f(t^*) - f^*] \prod_{i=1}^{m-1} u_N\left(t^i\right) = k_m(t,f),$$

and so

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}\left(\mathcal{G}_1, F\right)\right] = \int_{\mathcal{T}_m} \int_{\mathbb{R}^X} \tilde{w}_N(t,f)\, \kappa_m(dt,df) = ||\tilde{w}_N||_{\kappa_m}.$$

That is, $w_N = u_N$ *a.e.* in κ_m for all m. The next step is to show that $w_N = u_N$ *a.e.* in λ, and hence on each point on A. This fact will be proven by exhibiting a sequence of measures with respect to which w_N and u_N are almost everywhere equal, leading to the conclusion that $w_N = u_N$ *a.e.* in λ.

Define ν_m as the natural extension of λ to trajectories of length m,

$$\nu_m\left(dt\right) = \prod_{i=1}^{m} \lambda\left(dt^i\right).$$

Also, define $\tilde{\kappa}_m$ as the marginal of κ_m with objectives integrated out,

$$\tilde{\kappa}_m(dt) = \int_{\mathbb{R}^X} \kappa_m(dt,df) = \nu_m(dt) \int_{\mathbb{R}^X} k_m(t,f) \mathbb{P}_F(df).$$

Because $w_N = u_N$ *a.e.* in κ_m and w_N and u_n do not depend on the objective f for a set of full measure in κ_m, $w_N = u_N$ *a.e.* in $\tilde{\kappa}_m$. If ν_m is absolutely continuous with respect to the measure $\tilde{\kappa}_m$, then $w_N = u_N$ *a.e.* in ν_m. Absolute continuity holds if in turn

$$\int_{\mathbb{R}^X} k_m(t,f)\, \mathbb{P}_F(df) > 0 \quad a.e.\, \text{in}\, \nu_m.$$

The left hand side can be rewritten as

$$\int_{\mathbb{R}^X} k_m(t,f)\, \mathbb{P}_F(df) = \left(\prod_{i=1}^{m-1} u_N\left(t^i\right)\right) \mathbb{E}_{\mathbb{P}_F}\left[F\left(t^*\right) - F^*\right].$$

Recall that $\mathbb{P}_F$ has a no universal minimum, and so $u_N(x) > 0$ for all x and $\mathbb{E}_{\mathbb{P}_F}\left[f\left(t^*\right) - f^*\right] > 0$. Thus $w_N = u_N$ almost everywhere in ν_m. So $\hat{w}_N(x,t,y) = \hat{u}_N(x,t,y)$ for all $x \in A$ with t and f arbitrary, recalling that $\hat{u}_N(x,t,y) = \hat{u}_N(x)$.

The set A used to define λ was arbitrary, therefore $\hat{w}_N(x,t,y) = \hat{u}_N(x,t,y)$ for all $x \in X$ by letting A range over all countable sets in $\mathcal{B}_\tau$. To become convinced of this fact, note that any particular inputs x and t are contained entirely in some countable set A, and the values of y are determined by $\mathbb{P}_F$ independently of A. Finally, since $\hat{w}_N = \hat{u}_N$ for all N, then letting $N \to \infty$, it holds that $\hat{w}(x,t,y) = \hat{u}(x,t,y)$ for all $x \in X$, i.e. $\mathbb{P}_F$ is path uncorrelated. □

This subsection and the previous one have established necessary conditions for NFL; the next section shows that with slight additions, these conditions are also sufficient.

9.3.3 Sufficient Conditions for NFL

This section contains two theorems. The first theorem gives sufficient conditions for a function prior to be strongly NFL. The conditions are that the function prior must be path independent and identically distributed at every point. The second theorem gives sufficient conditions for a function prior to be weakly NFL in any additively decomposable performance criterion that is finitely uniquely dependent on the error sequence. Together with the results of the last two subsections, these two theorems prove that NFL priors are exactly those priors that defeat any attempts at learning. These results are summarized in the next section as the NFL Identification Theorems.

Theorem 9.3.5. *Any function prior that is path independent and identically distributed at every point is strongly NFL on* $\mathcal{O}_{\text{tr}}$.

Proof. Let $\mathbb{P}_F$ be identically-distributed and path independent. Fix $m \leq |X|$ and let $\mathcal{G} \in \mathcal{O}_{\text{tr}}$ such that $T_m < \infty$ $\mathcal{G}_f$*-a.s.* Fix $A \in \mathcal{B}[\mathbb{R}^m]$, a Borel set in $\mathbb{R}^m$. Referring to Equation 9.7, observe that

$$\mathbb{P}\left(F(Z_{T_j})_{j=1}^m \in A\right) = \int_A \sum_{n=1}^\infty \int_{H_n} \prod_{i=1}^n \mathcal{G}[t_1^{i-1}, y_1^{i-1}](dt^i)\mathbb{P}_F\left(\left(F\left(t^{T_j(t)}\right)\right)_{j=1}^M \in dy\right). \tag{9.34}$$

Next, note that because $\mathbb{P}_F$ is path independent,

$$\mathbb{P}_F\left(\left(F\left(t^{T_j(t)}\right)\right)_{j=1}^{M} \in dy\right) = \prod_{j=1}^{M} \mathbb{P}_F\left(F\left(t^{T_j(t)}\right) \in dy_j\right). \tag{9.35}$$

Also, because $\mathbb{P}_F$ is identically distributed, for any $x_0 \in X$,

$$\mathbb{P}_F\left(\left(F\left(t^{T_j(t)}\right)\right)_{j=1}^{M} \in dy\right) = \mathbb{P}_F\left(F\left(x_0\right) \in dy_j\right)^M. \tag{9.36}$$

Thus,

$$\begin{aligned}\mathbb{P}\left(F(Z_{T_j})_{j=1}^{m} \in A\right) &= \int_A \mathbb{P}_F\left(F\left(x_0\right) \in dy_j\right)^M \sum_{n=1}^{\infty} \int_{H_n} \prod_{i=1}^{n} \mathcal{G}[t_1^{i-1}, y_1^{i-1}](dt^i) \\ &= \int_A \mathbb{P}_F\left(F\left(x_0\right) \in dy_j\right)^M. \end{aligned} \tag{9.37}$$

Since this probability is independent of $\mathcal{G}$, the distribution of $\left(F\left(Z_{T_j}\right)\right)_{j=1}^{m}$ is a constant depending on m for any $\mathcal{G}$ such that $T_m < \infty$ $\mathcal{G}_f$-*a.s.* That is, $\mathbb{P}_F$ is strongly NFL. □

Theorem 9.3.6. *Any function prior that is mean-constant and path uncorrelated is weakly NFL on $\mathcal{O}_{\text{tr}}$ over all additively decomposable and finitely uniquely dependent performance criteria.*

Proof. Let $\mathbb{P}_F$ be mean-constant and path uncorrelated. Let ϕ be a performance criterion that is additively decomposable and finitely uniquely dependent on the error sequence up to time $M \leq |X|$. Let $\mathcal{G} \in \mathcal{O}_{\text{tr}}$ such that $T_M < \infty$ $\mathcal{G}_f$-*a.s.* Note that

$$\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[\sum_{j=1}^{M} w_j E_{T_j}^f\right]$$

for some sequence (w_j) with $w_M > 0$ and the error process $E_{T_j}^f$. Let H_n^M be the set of stopping trajectories for T_M of length n. Taking the expectation over $\mathbb{P}_F$ on both sides and decomposing the sum,

$$\mathbb{E}_{\mathbb{P}_F}\left[\phi(\mathcal{G}, f)\right] = w_M \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}f}\left[F(Z_{T_M}) - F^*\right] + \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}f}\left[\sum_{j=1}^{M-1} w_j E_{T_j}^f\right] = \xi(M).$$

The proof follows if $\xi(M)$ does not depend on $\mathcal{G}$, which can be proven by induction on M.

For the base case, it holds that $T_1 = 1$, and so for arbitrary x_0,

$$\begin{aligned} \xi(1) &= \int_X \mathcal{G}[\emptyset, \emptyset](dx) \int_{\mathbb{R}} w_1 y_1 \mathbb{P}_F\left(F(x) \in dy\right) - \int_{\mathbb{R}} w_1 y^* \mathbb{P}_F\left(F^* \in dy^*\right) \\ &= w_1\left[\mathbb{E}_{\mathbb{P}_F}[F(x_0)] - \mathbb{E}_{\mathbb{P}_F}[F^*]\right] = w_1 C, \end{aligned} \tag{9.38}$$

where C is a constant determined by the constant mean of the prior and the expected minimum. Thus the base case is satisfied.

To extend this result to arbitrary M, suppose for the induction hypothesis that

$$\xi(M-1) = C \sum_{j=1}^{M-1} w_j, \tag{9.39}$$

and it will be shown that $\xi(M) = w_M C + \xi(M-1)$, which is independent of $\mathcal{G}$. Consider the conditional expectation

$$\mathbb{E}_{\mathbb{P}_F}\left[F(z_M) \mid F(z_j),\, j = 1, \dots, M-1\right] = g(z_1, \dots, z_M). \tag{9.40}$$

By the properties of conditional expectations,

$$\mathbb{E}_{\mathbb{P}_F} \mathbb{E}_{\mathcal{G}f}\left[F(Z_{T_M})\right] = \mathbb{E}_{\mathbb{P}_F} \mathbb{E}_{\mathcal{G}f}\left[g\left(Z_{T_1}, \dots, Z_{T_M}\right)\right]. \tag{9.41}$$

But $Z_{T_1}, \dots, Z_{T_M}$ are distinct, and $\mathbb{P}_F$ is path uncorrelated and mean-constant, so for arbitrary x_0,

$$g\left(Z_{T_1}, \dots, Z_{T_M}\right) = \mathbb{E}_{\mathbb{P}_F}\left[F(Z_{T_M})\right] = \mathbb{E}_{\mathbb{P}_F}\left[F(x_0)\right] = C + \mathbb{E}_{\mathbb{P}_F}\left[F^*\right]. \tag{9.42}$$

Therefore,

$$\mathbb{E}_{\mathbb{P}_F} \mathbb{E}_{\mathcal{G}f}\left[F(Z_{T_M}) - F^*\right] = C \tag{9.43}$$

and by definition, $\xi(M) = C + \xi(M-1)$ as required. The optimizer $\mathcal{G}$ was arbitrary, and so $\mathbb{P}_F$ is weakly NFL in ϕ on $\mathcal{O}_{\mathrm{tr}}$, completing the proof. ☐

The previous two theorems have shown that function priors whose paths are not correlated with themselves are necessarily subject to NFL. These results can be combined with the results of the previous section to give necessary and sufficient conditions for the strong and weak NFL properties.

9.3.4 NFL Identification Theorem

The results of the previous three subsections can be aggregated into a pair of theorems named the *NFL Identification Theorems*, which give necessary and sufficient conditions for a function prior to possess the strong or weak NFL properties. In the case of the weak NFL property, these theorems only apply to additively decomposable performance criteria that are finitely uniquely dependent on the error sequence. Among the performance criteria presented in Chapter 7, only ζ_{T_m} and ϕ_T satisfy this criteria. Because of the nature of the weak NFL property, the sufficient conditions for weak NFL always depend upon the specific performance criteria to which it is applied.

The NFL Identification Theorems expand the application of the NFL properties to the more general setting of arbitrary measure spaces. Rowe et al. previously showed that the NFL property applies in infinite settings, but their results were limited to uniform priors over c.u.p. subsets [171]. The next two theorems give for the first time an exact characterization of what it means for an arbitrary function prior to be subject to the NFL property for trajectory-restricted optimizers.

Theorem 9.3.7. Strong NFL Identification Theorem. *A function prior $\mathbb{P}_F$ is strongly NFL on $\mathcal{O}_{\text{tr}}$ if and only if $\mathbb{P}_F$ is identically distributed at each point and path independent.*

Proof. Strong NFL $\implies$ identically distributed and path independent: Suppose $\mathbb{P}_F$ is strongly NFL on $\mathcal{O}_{\text{tr}}$. Lemma 9.3.1 shows that $\mathbb{P}_F$ is identically distributed at every point, so it only remains to show that $\mathbb{P}_F$ is path independent.

First, suppose $\mathbb{P}_F$ has a universal minimum. Then $\mathbb{P}_F$ is universally constant by Lemma 9.2.2. But if $\mathbb{P}_F$ is universally constant, then it is trivially path independent.

If $\mathbb{P}_F$ has no universal minimum, then Theorem 9.3.4 implies that $\mathbb{P}_F$ is path independent, since $\mathbb{P}_F$ is weakly NFL in ζ_{T_m} by Theorem 9.2.1.

Path independent and identically distributed $\implies$ Strong NFL: This result was proven as Theorem 9.3.5. □

Theorem 9.3.8. Weak NFL Identification Theorem. *A function prior* $\mathbb{P}_F$ *is weakly NFL on* $\mathcal{O}_{\mathrm{tr}}$ *over all additively decomposable performance criteria that are finitely uniquely dependent on the error sequence if and only if* $\mathbb{P}_F$ *is mean-constant and path uncorrelated.*

Proof. Weak NFL $\implies$ *mean-constant and path uncorrelated:* Suppose $\mathbb{P}_F$ is weakly NFL on $\mathcal{O}_{\mathrm{tr}}$ and ζ_{T_m} for all m. Lemma 9.3.2 shows that $\mathbb{P}_F$ is mean-constant, so it only remains to show that $\mathbb{P}_F$ is path independent.

First, suppose $\mathbb{P}_F$ has a universal minimum. Then $\mathbb{P}_F$ is universally constant by Lemma 9.2.2. But if $\mathbb{P}_F$ is universally constant, then it is trivially path uncorrelated.

If $\mathbb{P}_F$ has no universal minimum, then Theorem 9.3.4 immediately implies that $\mathbb{P}_F$ is path uncorrelated.

Path uncorrelated and mean-constant $\implies$ *Weak NFL:* This result was proven as Theorem 9.3.6. □

9.4 Conclusion

In this chapter, an array of NFL Theorems were presented, leading ultimately to the discovery of the NFL Identification Theorems. These theorems expand the applicability of NFL to all trajectory-restricted optimizers in arbitrary measure spaces. They also provide sufficient conditions that demonstrate what causes a function prior to induce the NFL property. Given the importance ascribed to the original NFL theorems, these accomplishments should be of interest on their own.

The fact that the strongly NFL priors must be path-independent discredits the hypothesis that NFL prevents general-purpose optimizers, since such a claim is tantamount to saying that the general prior over real-world problems has no internal structure whatsoever. NFL implies that a function prior is unlearnable, yet it would seem strange to assume that physical reality is unlearnable. In relatively small, discrete domains it may well be true that no objective is more likely than any other. For any large, realistic domain, however, there is substantial internal structure to any general prior that is not just problem-specific. Concrete examples of very general priors not subject to NFL will be given in the next chapter. In spite of NFL, general-purpose learning should be possible. These ideas are explored further in the next chapter.

Even if certain interpretations of NFL are somewhat discredited, there remains a large circle of ideas related to NFL that are as true and relevant as ever. If the practitioner is only interested in a small subset of objectives, it is almost a truism that a tailored optimization algorithm should outperform a general-purpose optimizer. Furthermore, even if some optimizers are better than others, it does not follow that there is a unique best optimizer, even for a relatively focused function prior. The performance of an optimizer is intimately linked to the conditions under which it is tested, and any random test procedure corresponds to some function prior. All of these ideas together may be regarded as the NFL way of thinking, and this paradigm is still useful and productive, as will be shown in the next chapter.

Chapter 10

The Geometry of Optimization and the Optimization Game

This chapter examines the philosophical and theoretical implications of the NFL results from the previous chapter. It introduces a duality between optimizers and function priors that formalizes Wolpert and Macready's notion of alignment within a performance-based geometry for the space of optimizers. Optimization is then explored as a game-theoretic exercise, pitting an optimizer against a function prior. The possibility of optimizing the optimization process is considered, and it is conjectured that the information generated by the optimization process should play a key role. These results lead to the formulation of the information-maximization principle for static optimization. An optimizer implementing this principle, evolutionary annealing, will be introduced in Chapter 11. The ideas in this chapter suggest a number of points of departure for further research and inquiry, all opened up by the rigorous formal approach adopted in this dissertation.

10.1 The Reality Prior

In practice, an objective function corresponds to some observable quantity that has a utility for someone, such as the amount of oil recovered from a wellhead, the number of errors made by an automated manufacturing system, the purity of a chemical substance, or the profit on a financial trade. What sort of function prior governs objectives derived from reality? Does the nature of such a prior differ if only certain kinds of quantities are considered? If all such objectives are considered in aggregate, is the resulting prior subject to NFL? These questions are considered briefly in this section, with the conclusion that a prior can be quite general without inducing NFL. The term *reality* *prior* will be used for the overarching prior governing all reality-derived objectives. There is good reason to speculate that the reality prior in fact does not obey NFL and instead contains internal regularities that make general learning possible. Such speculation is related to conjectures proposed in the context of inductive inference that connect the universal prior to program description lengths via Occam's razor. Before taking up this topic, this inquiry begins with a discussion of the frequency of NFL priors.

10.1.1 The Frequency of NFL Priors

Now that the NFL priors have been completely identified, it is possible to pose the question of how common strongly NFL priors may be. As it happens, far from being in the vast majority, NFL priors are extraordinarily rare. As mentioned in Chapter 9, Igel and Toussaint [100] previously showed that a vanishingly small portion of function subsets possess the NFL property under a uniform prior. Now that Theorem 9.3.7 has identified the strongly NFL priors completely, it can be stated in general that Igel and Toussaint's results hold for all priors, not just c.u.p. subsets. NFL priors are not just rare, they have measure zero even in finite search spaces.

Suppose the search space X is finite, and that objective functions taking values from a finite space Y are assumed to have probability one. The effective objective function space then has size $D = |Y|^{|X|}$. Since this function space is finite, then the space of priors over it is a finite-dimensional vector space isomorphic to $\mathbb{R}^{D-1}$. To convert a probability vector in $\mathbb{R}^{D}$ to $\mathbb{R}^{D-1}$, use spherical coordinates, remove the radius, and expand the remaining coordinates from $[-\pi, \pi)$ to all of $\mathbb{R}$ by the transformation $g(x) = x/\pi(x+1)$. Under this transformation, the function priors on Y^X cover $\mathbb{R}^{D-1}$ completely. Strongly NFL priors are a proper subset of the identically distributed priors. In order for a prior to be identically distributed, if must have the same distribution at each input. It should be clear that the set of identically distributed priors is isomorphic to $\mathbb{R}^{|X|-1}$ under a similar transformation as for $\mathbb{R}^{D-1}$. As long as $|Y| > 1$, it follows that $|X| < |D|$, and thus the set of identically distributed priors has Lebesgue measure zero in $\mathbb{R}^{D-1}$. The set of strongly NFL priors is even smaller and therefore has measure zero as well. Thus if an infinite sequence of priors were chosen at random, with probability one, not a single one would be identically distributed, much less NFL!

The set of NFL priors can be made larger by considering only priors that take on discrete and finite probabilities from a set $M \subseteq [0, 1]$. This sort of argument might be made from an axiomatic assumption that the world is computable, rejecting the theory of a continuum. Even so, the set of NFL priors would grow exponentially small as a M, $|X|$, or $|Y|$ are increased. Thus, on purely statistical basis, one does not expect to encounter an NFL prior, even if there are finitely many priors.

The sparsity of NFL priors is not in itself a reasonable objection to the hypothesis that the reality prior is strongly NFL. Even though the space of priors is large, most priors are chaotic and random and lack compact descriptions. By contrast, an NFL prior has a short and simple description and should therefore be preferred over priors with long description lengths on the principle of Occam's Razor. However, this argument can be turned against NFL. As a prior, it has short descriptions, but functions drawn randomly from a non-constant NFL prior typically have long descriptions. In an infinite domain, such a function can only be described on average by listing its value at an infinite number of points. Further, experience in the real world suggests that a correct prior should prefer abundant substructure, especially local regularity and globally repeated structure (decomposability). Such a prior may have a longer description length than an NFL prior, but the functions drawn from it would have shorter descriptions, balancing the complexity of both the prior and the objectives it prefers.

10.1.2 Diffusion Prior

The previous section proposed that one should expect the reality prior to possess more regularity than a strongly NFL prior does. But do general priors with regular structure exist? In this section, the diffusion prior will be discussed as an example of a prior satisfying one of the desired qualities in a prior, that of local regularity. The reality prior likely possesses more regularity and structure than a diffusion prior; this example is only intended to show that such regularities exist within general-purpose priors.

A diffusion prior may be defined as a function prior over objectives on subsets of $\mathbb{R}^n$ based on the Brownian motion. The Brownian Motion assumes random expansion in space at a rate equal to the square root of the time elapsed. It is a well-studied mathematical object that plays a role in mathematical theories ranging from physics to finance [38, 105]. The Brownian motion can be defined as a prior over all continuous functions, or even as a prior over discontinuous ones. However, it enforces a constraint that if a function has a certain value at one point, then it is likely (but not required) to have similar values at nearby points.

The standard Brownian Motion is a random process in one variable, often denoted by $W = (W_t)_{t\in[0,\infty)}$, such that the increments of the Brownian Motion are normally distributed with a variance the size of the increment, i.e.

$$W_t - W_s \sim \mathcal{N}\left(W_s, t-s\right), \ \forall s < t.$$

Typically, the process is started at $W_0 = 0$ for convenience, but the process can be initialized at any point. The phrase “Brownian Motion started at x” indicates that $W_0 = x$.

The Brownian Motion can be constructed as a prior over continuous functions ($C[0,\infty)$) against a σ-algebra of cylinder sets (see e.g. Karatzas and Shreve [105]). Thus the distribution of the Brownian Motion is a function prior. A function drawn from this Brownian prior is a continuous trajectory of the Brownian Motion.

The Brownian prior has constant mean ($\mathbb{E}W_t = W_0$), but it is not identically distributed at each point and is path dependent. Thus such a prior is non-NFL. It is easy to demonstrate this fact because W_t is a martingale and so

$$\mathbb{E}\left[W_t \mid W_s\right] = W_s \neq W_0 = \mathbb{E}W_t \text{ for } 0 < s < t.$$

As a non-NFL prior, it follows that some optimizers perform better than others on this prior. This fact is also easily demonstrated. Consider the search space given by $t \in [0,1]$. Let $\mathcal{G}_1$ be a deterministic optimizer that proposes evenly spaced points in order, say,

$$0, \frac{1}{100}, \frac{2}{100}, \frac{3}{100}, \frac{4}{100}, \frac{5}{100}, \ldots,$$

and let $\mathcal{G}_2$ be another deterministic optimizer that iteratively splits $[0,1]$ with evenly spaced points, i.e

$$0, 1, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8}, \frac{1}{16}, \ldots$$

$\mathcal{G}_2$ has better performance than $\mathcal{G}_1$ on a Brownian prior, because at each step it eliminates a larger proportion of functions from consideration than $\mathcal{G}_2$ by placing more constraints on the available functions.

Define an n-dimensional diffusion prior as a functional of one or more Brownian Motions on $\mathbb{R}^n$. It can be described as a random objective Y indexed by $x \in [0,1]^n$ given by

$$Y_x = g(W_{1,x_1}, \ldots W_{n,x_n}),$$

where $W_1, \ldots, W_n$ are n independent Brownian Motions, and g is a Borel-measurable function from $\mathbb{R}^n$ to $\mathbb{R}^n$.

Let $\mathbb{P}_Y$ be the distribution of Y. Consider the search space $[0,1]^n$. Then a sample from $\mathbb{P}_Y$ is a function over the search space. This prior can be extended to all of $\mathbb{R}^n$ by transforming the space. The diffusion prior $\mathbb{P}_Y$ is a functional of the Brownian Motion, and as long as the function g is non-degenerate, $\mathbb{P}_Y$ is not strongly NFL.

Readers familiar with the use of Gaussian processes in predictive function modeling should notice the similarity. The function g defining the diffusion prior corresponds to the choice of kernel in a Gaussian process. The driving Brownian Motions could also be shifted backwards in time by an arbitrary amount (e.g. $\tilde{W}_{n,t} = W_{n,t+\sigma_0^2}$) to obtain a non-degenerate initial distribution. With this in mind, an optimizer could perform well on a diffusion prior by choosing evaluation points in such a way as to minimize the conditional variance of the objective function given the evaluation points under the kernel g. Because of this relationship, a diffusion prior might also be called a Gaussian prior.

The diffusion prior provides an example of a non-NFL prior that can place positive probability on all subsets of continuous functions. As this example proves, NFL does not imply that successful learning is only possible on specific problems.

10.1.3 The Universal Prior

The previous section proved that very general non-NFL priors exist. But what sort of prior is the reality prior? Some conjectures have been previously offered on this topic in the context of inductive inference [37, 47, 98, 178, 190, 191]. Ideas concerning a universal prior have typically centered on the modernized version of Occam's razor, which says that when one is faced with competing hypotheses that explain some data, the simplest hypothesis is most likely to be correct.

This research is rooted in the idea that the universe is generated by a computable program, an idea that goes back to Zuse, and even further back to Leibniz [178]. Given that the universe has an observed state x, then according to Occam's razor, the most likely program computing the universe is the shortest program that computes x. In 1964, Solomonoff [190, 191] proposed a universal measure over bit sequences conforming to this principle, given by

$$\mathbb{P}'_M(x) = \sum_{\substack{\text{program prefixes}\, p \\ \text{that compute}\, x}} 2^{-|p|}. \tag{10.1}$$

Similar measures with a basis in information theory were subsequently proposed by Chaitin and Cover that model the entropy of the observation [37, 47].

Solomonoff's measure is enumerable but not computable because of the halting problem [178, 191]. Computable variants have been proposed based on Minimum Description Length and Kolmogorov complexity [47, 113, 178]. Most recently, Schmidhuber [178] proposed the Speed Prior in 2002 after work by Hutter [98] exhibited an algorithm to enumerate all programs that produce an output prefix after a fixed of number of steps. The Speed Prior is similar to Solomonoff's measure in spirit, but accounts for the computation time as well, making it computable in the limit. Schmidhuber provided an algorithm for computing the Speed Prior in finite time within a given tolerance.

Viewed as probabilities over objectives (as program subroutines), neither Schmidhuber's nor Solomonoff's measures could possibly be strongly NFL, nor could any similar measure that prioritizes observations on the basis of compressibility. NFL requires that the future be incomputable on the basis of the past, and compressible programs necessarily encode computable regularities. It seems likely that formal results could be derived to demonstrate this claim. Such a result is left as future work, discussed again briefly in Chapter 14. However, if the reality prior possesses the NFL property, then any universal measure based on Occam's razor must be false, since NFL assumes that complex programs are substantially more common than simple ones.

10.1.4 The Meaning of NFL

The discovery that NFL implies path independence (Section 9.3) makes it possible to understand exactly what NFL means. Whenever NFL holds, learning is impossible. The past is irrelevant to the future, and there is no means of predicting the outcome of any action. If the reality prior is NFL, then anyone betting that the sun will rise tomorrow is taking a substantial risk. Under NFL, one could go to sleep at night on Earth, and wake up in the morning to find himself in the middle of Alpha Centauri with a few extra appendages. NFL admits no rational basis for making any decisions or forming any definite opinion about any future detail, no matter how immediate or trivial.

The assumption that the reality prior is NFL can be defeated simply by observing the existence of humans. People can and do predict numerous aspects of the future. At a mundane level, if a person places an object in a room and leaves, then if no one else enters the room, the person is bound to find the object in the place where he left it when he returns. In essence, the laws of physics are nothing other than a simple model of future physical interactions. The very fact that such laws can be stated in a highly compressed form with reliable predictive power implies directly that the general structure of the universe is path dependent. The reality prior is not NFL. There exist general-purpose optimizers that outperform others on all tasks averaged according to their likelihood.

10.2 Duality and the Geometry of Optimization

The existence of performant general-purpose optimizers does not preclude specialized optimizers from performing better on specific problem classes. Indeed, there is a natural pairing between problem classes and the optimizers that perform optimally on them. Wolpert and Macready's concept of alignment between optimization algorithms and function priors can be made rigorous and formal by expressing this relationship as a duality, a non-degenerate bilinear mapping over optimizers and priors. This duality forms the basis for assessing the performance of optimizers over particular problem classes.

10.2.1 Duality Based on Average Error

The duality results in this section relate the space of long-running trajectory-restricted generalized optimizers $\mathfrak{A}[\mathcal{X}]$ from Section 6.2 for some $\mathcal{X} \subseteq \mathcal{MF}$ to the space of admissible function priors $\mathcal{M}_a[\mathbb{R}^X]$ from Section 9.2.1. For any performance criterion ϕ, a bilinear mapping can then be given by

$$\langle \mathcal{G}_f, \mathbb{P}_F \rangle_\phi = \mathbb{E}_{\mathbb{P}_F}\left[\phi\left(\mathcal{G}, F\right)\right]. \tag{10.2}$$

It is important to note that this mapping is linear over the vector structure of $\mathfrak{A}\left[\mathcal{X}\right]$ rather than that of $\mathcal{X}$, since the mapping is actually non-linear over $\mathcal{X}$.

Proposition 10.2.1. *For any fixed performance criterion ϕ, $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_\phi$ is a bilinear mapping on $\mathfrak{A}\left[\mathcal{X}\right] \times \mathcal{M}\left[\mathbb{R}^X\right]$ for any $\mathcal{X} \subseteq \mathcal{MF}$.*

Proof. Note that

$$\langle \mathcal{G}_f, \mathbb{P}_F \rangle_\phi = \mathbb{E}_{\mathbb{P}_F}\mathbb{E}_{\mathcal{G}F}\left[h(Z)\right]$$

for some function h dependent on ϕ. The result is then a trivial consequence of the linearity of the integral. □

To obtain a duality, this bilinear mapping must be *non-degenerate*. That is, if $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_\phi$ is zero for all $\mathcal{G}_f$, then $\mathbb{P}_F$ must be the zero measure. And if $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_\phi$ is zero for all $\mathbb{P}_F$, then $\mathcal{G}_f$ must be identically zero.

Notably, $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is degenerate on universally constant priors. If $\mathbb{P}_F$ is a universally constant prior, then $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}} = 0$ for all $\mathcal{G}_f$ because every point is optimal. But there are many universally constant priors that are non-zero. So universally constant priors must be excluded to reach a duality.

Notice that the vector sum or scalar product of two distinct universally constant priors is universally constant. That is, if $F = c$ and $G = d$ are two universally constant priors, then $F+G = c+d$ is a constant, and so is $\alpha F = \alpha c$. So universally constant priors form a vector subspace of $\mathcal{M}_a[\mathbb{R}^X]$. Let $\mathcal{UC}$ be the vector subspace of universally constant priors, and define $\mathcal{NC} = \mathcal{M}_a[\mathbb{R}^X] \perp \mathcal{UC}$ to be the vector subspace of function priors with universally constant priors removed.

Theorem 10.2.2. *The bilinear mapping* $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ *is non-degenerate over* $\mathfrak{A}\left[\mathcal{MF}_{\mathrm{tr}}\right] \times \mathcal{NC}$ *if* $|X| \geq m+1$, *and therefore the vector space of optimizers* $\mathfrak{A}\left[\mathcal{MF}_{\mathrm{tr}}\right]$ *and the vector space of function priors* $\mathcal{NC}$ *are in duality under this mapping.*

Proof. **Non-degeneracy in $\mathcal{NC}$.** Assume $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is zero for all $\mathcal{G}_f \in \mathfrak{A}\left[\mathcal{MF}_{\mathrm{tr}}\right]$, and that $\mathbb{P}_F$ is not zero. Then $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is zero for $\mathcal{A}^x[t,f] = \delta_x$ for any $x \in X$. Thus

$$\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}} = \left\langle \mathcal{A}^x_f, \mathbb{P}_F \right\rangle_{\zeta_{T_m}} = \mathbb{E}_{\mathrm{P}_F}\left[F(x) - F^*\right] = 0.$$

Since x was arbitrary, $F(x) = F^*$ almost surely in $\mathbb{P}_F$ for all x, and so $\mathbb{P}_F$ is universally constant, which contradicts the fact that $\mathbb{P}_F \in \mathcal{NC}$. Therefore $\mathbb{P}_F$ is zero.

Non-degeneracy in $\mathfrak{A}\left[\mathcal{MF}_{\mathrm{tr}}\right]$. Nondegeneracy of optimizers will be shown by using the pigeonhole principle. Assume $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is zero for all $\mathbb{P}_F \in \mathcal{NC}$ for some fixed, nonzero $\mathcal{G}_f$. The goal is to construct a prior $\mathbb{P}_F$ that cannot be perfectly decided by a trajectory-restricted optimizer. To this end, choose $m+1$ distinct points $x_1, \ldots, x_{m+1} \in X$, which is possible since $|X| \geq m+1$.

Now we will construct a combination of $m+1$ function priors that cannot be distinguished by evaluating just m points. Let $G_1, \ldots, G_{m+1}$ be these function priors. Set $G_i(x_i) = \delta{-1}$ for all $i \in \{1, \ldots, m+1\}$ so that $G_i^* = -1$. For $y \neq x_i$, let each G_i have an exponential distribution over the nonnegative numbers, $\mathbb{P}(G_i(y) \in dx) = 1_{[0,\infty)} e^{-x}\, dx$. Define G so that

$$\mathbb{P}_G = \frac{1}{m+1} \sum_{i=1}^{m+1} \mathbb{P}_{G_i}, \tag{10.3}$$

meaning that $G(x)$ is sampled by first choosing one of the G_i uniformly at random and then sampling $G_i(x)$. Each of the G_i is $\mathcal{B}[\mathbb{R}^X]$-measurable, and thus G is as well. It should be clear that $G^* = -1$, and that for any $y \in x$ such that $y \neq x_i$ for all i, $G(y) \geq 0$ with probability one. In order to have $\langle \mathcal{G}_f, \mathbb{P}_G \rangle_{\zeta_{T_m}} = 0$, $\mathcal{G}_f$ must determine which of the G_i was actually sampled, since it must hold that $G(Z^*_{T_m}) - G^* = 0$ almost surely. This fact implies $Z^*_{T_m} = x_i$ for the value of i selected randomly by G. So $\mathcal{G}$ perfectly distinguishes all $m+1$ cases after just $m-1$ evaluations, proposing x_i at or before the m^{th} evaluation. Consequently, $\mathcal{G}$ must use some mechanism other than function evaluation to identify which of the $m+1$ functions it is handling. That is, $\mathcal{G}$ is not trajectory-restricted, in contradiction to the fact that $\mathcal{G} \in \mathcal{MF}_{\text{tr}}$. Thus $\mathcal{G}_f$ is zero, and $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is non-degenerate. $\square$

Thus the spaces of non-constant function priors and the space of trajectory-restricted optimizers are dual vector spaces. The nature of this duality is that of posterior and prior. This fact can be seen more clearly by stating the joint distribution explicitly. Suppose $Z \sim \mathcal{G}_F$, and then $\mathcal{G}_F(A) = \mathbb{P}(Z \in A \mid F)$. Then

$$\mathcal{G}_F(A)\, \mathbb{P}_F(B) = \mathbb{P}(Z \in A \mid F \in B)\, \mathbb{P}(F \in B) = \mathbb{P}(Z \in A, F \in B),$$

and using the set of stopping trajectories $H = \bigcup_n H_n$ from Equation 6.8 to remove T_m,

$$\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}} = \mathbb{E}\left[F\left(Z^*_{T_m}\right) - F^*\right] = \int_{\mathbb{R}^X \times H} f(t^*) - f^* \; \mathbb{P}_{Z,F}(dt, df).$$

That is, an optimizer's performance on a function prior under ζ_{T_m} is just the average error over the joint distribution of Z and F. An optimizer and a function prior together form a system with a well-defined performance given by $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$. Given a joint distribution over trajectories and objectives, the decomposition into optimizer and function prior is unique. In addition, every optimizer-prior system has an alternate decomposition as a prior over trajectories and a posterior over objectives given trajectories, i.e.

$$\mathbb{P}(Z \mid F)\, \mathbb{P}(F) = \mathbb{P}(F \mid Z)\, \mathbb{P}(Z).$$

This alternate system may be thought of as a solution to the complementary problem of finding the function that minimizes the error of a given trajectory.

The duality between $\mathfrak{A}\left[\mathcal{O}_{\mathrm{tr}}\right]$ and $\mathcal{NC}$ introduced in this subsection formalizes the idea of geometric alignment between problems and priors advanced by Wolpert and Macready. However, this duality was restricted to the performance criterion ζ_{T_m}. The next subsection explores how this concept can be generalized to other performance criteria.

10.2.2 Duality Under Arbitrary Performance Criteria

Although $\left\langle \mathcal{G}_f, \mathbb{P}_F \right\rangle_\phi$ is bilinear over all of $\mathfrak{A}[\mathcal{MF}] \times \mathcal{M}_a[\mathbb{R}^X]$, it may be degenerate on different subspaces due to the nature of the performance criterion. For ζ_{T_m}, degeneracy could be induced by either a non-constant prior or an optimizer with access to more than just the function evaluations along the trajectory. In general, a duality can be obtained on a fixed non-trivial performance criterion ϕ by eliminating the null space of both arguments.

Definition 10.2.1. *The optimizer null space of a performance criterion ϕ on an optimizer space $\mathfrak{A}[\mathcal{X}]$ is the set of function priors for which ϕ is zero for all optimizers in $\mathcal{X}$, denoted*

$$\mathcal{N}_\phi[\mathcal{X}] = \{\mathbb{P}_F \in \mathcal{M}_a[\mathbb{R}^X] \;\mid\; \forall \mathcal{G}_f \in \mathfrak{A}[\mathcal{X}], \left\langle \mathcal{G}, F \right\rangle_\phi = 0\}. \tag{10.4}$$

Definition 10.2.2. *The prior null space of a performance criterion ϕ on a space of function priors $\mathcal{P}$ is the set of optimizers for which ϕ is zero for all priors in $\mathcal{P}$, denoted*

$$\mathcal{N}^\phi[\mathcal{P}] = \{\mathcal{G} \in \mathfrak{A}[\mathcal{MF}] \;\mid\; \forall \mathbb{P}_F \in \mathcal{P}, \left\langle \mathcal{G}, F \right\rangle_\phi = 0\}. \tag{10.5}$$

A space of optimizers and a space of priors are in duality under a performance criterion if and only if the optimizer null space and the prior null space are excluded.

Proposition 10.2.3. *The space of optimizers $\mathfrak{A}[\mathcal{X}]$ and the space of function priors $\mathcal{P}$ are in duality on ϕ if and only if $\mathcal{N}_\phi[\mathcal{X}] \cap \mathcal{P} = \{0\}$ and $\mathcal{N}^\phi[\mathcal{P}] \cap \mathfrak{A}[\mathcal{X}] = \{0\}$.*

Proof. Suppose that $\mathcal{N}_\phi[\mathcal{X}] \cap \mathcal{P} = \{0\}$ and $\mathcal{N}^\phi[\mathcal{P}] \cap \mathfrak{A}[\mathcal{X}] = \{0\}$. Then the definition of $\mathcal{N}[\cdot]$ implies that $\langle \cdot, \cdot \rangle_\phi$ is non-degenerate on $\mathcal{A}[\mathcal{X}] \times \mathcal{P}$, and duality follows from the bilinearity of $\langle \cdot, \cdot \rangle_\phi$. Conversely, if $\mathfrak{A}[\mathcal{X}]$ and $\mathcal{P}$ are in duality, non-degeneracy guarantees the desired result. $\square$

Without going into further depth, the optimizer space $\mathcal{O}_{\text{tr}}$ and the space of non-constant priors $\mathcal{NC}$ are also in duality under the performance criteria ϕ_w, ϕ_T, $\psi_\epsilon - 1$, and $1 - \sigma_\epsilon$ with non-trivial parameter assignments using the definitions in Section 7.1. Duality can also be obtained for many of these same performance criteria in a larger space of information-restricted optimizers provided that the amount of information obtained from each function evaluation can be bounded, as will be shown next.

10.2.3 Duality and Information Restrictions

The past several chapters have focused on the performance of trajectory-restricted optimizers. However, many of the same results also apply to more general information-restricted optimizers with some modifications. Keep in mind that information in information-restricted optimizers is distinct from the information contained in a filtration, although the two are related in that the information passed to an information-restricted optimizer generates a corresponding filtration under appropriate conditions.

As with $\mathcal{MF}_{\text{tr}}$, there is a duality between most optimizers in $\mathcal{MF}_{\text{ir}}$ and all function priors in $\mathcal{NC}$. The map $\langle \mathcal{G}_f, \mathbb{P}_F \rangle_{\zeta_{T_m}}$ is bilinear on all of $\mathcal{MF}$, so the only hindrance to duality is the degeneracy of certain optimizers. Some information-restricted optimizers are degenerate in this map even on $\mathcal{NC}$, because the information function can be used to pass information identifying the objective function back to the optimizer. As a result, it is impossible to use the pigeonhole principle to force non-degeneracy as in the proof of Theorem 10.2.1. If the search space X is large enough, this problem can be avoided by bounding the size of the information trajectory that can be returned.

Definition 10.2.3. *An optimizer $\mathcal{G} \in \mathcal{MF}_{\text{ir}}$ is of bounded information dimension if the length of trajectories returned by its information function is bounded above by some $M < \infty$, i.e. $|I(x, f)| \leq M < \infty$ for all x, f. In this case, $\mathcal{G}$ has information dimension bound M.*

Once the information dimension is bounded, non-degeneracy of $\left\langle \mathcal{G}_f, \mathbb{P}_F \right\rangle_{\zeta_{T_m}}$ can be proven for $\mathcal{MF}_{\text{tr}}$. Unfortunately, the information bound is not preserved when two information-restricted optimizers are convexly combined, since the convex combination depends on both information functions and thus has information dimension bound by $2M$, not M. However, the information dimension bound is preserved by convex combinations over optimizers with the same information function. Let I be an information function with information dimension $M < \infty$. Then $\left\langle \mathcal{G}_f, \mathbb{P}_F \right\rangle_{\zeta_{T_m}}$ is non-degenerate on $\mathcal{O}^I_{\text{ir}} \times \mathcal{NC}$.

Theorem 10.2.4. *Let* $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$ *be an information function with information dimension bound* $M < \infty$*. Then the bilinear mapping* $\left\langle \mathcal{G}_f, \mathbb{P}_F \right\rangle_{\zeta_{T_m}}$ *is non-degenerate over* $\mathfrak{A}\left[\mathcal{MF}^I_{\text{ir}}\right] \times \mathcal{NC}$ *if* $|X| \geq M(m-1)+2$*, and therefore the vector space of optimizers* $\mathfrak{A}\left[\mathcal{MF}^I_{\text{ir}}\right]$ *and the vector space of function priors* $\mathcal{NC}$ *are in duality under this mapping.*

Proof. Repeat the proof of Theorem 10.2.1 using $M(m-1)+2$ distinct points and function priots in the second half of the proof instead of just $m+1$ points and functions. In this case, with an information bound of M, an optimizer $\mathcal{G}_f \in \mathfrak{A}[\mathcal{MF}_{\text{ir}}]$ can only distinguish at most $M(m-1)+1$ possibilities, since the information returned to $\mathcal{G}$ has only $M(m-1)$ degrees of freedom. $\square$

As for $\mathcal{MF}_{\text{tr}}$, this same duality also holds for ϕ_w, ϕ_T, $\psi_\epsilon - 1$, and $1 - \sigma_\epsilon$ as well.

Duality is a fertile topic for further analysis, and this dissertation does not have the space to move beyond its immediate consequences. Nonetheless, the duality mapping explored in this section articulates the notion of alignment between optimizers and function priors, and provides a formal environment within which performance may be analyzed. The particular goal in this case is to identify the optimal optimizer for a particular test procedure or, conversely, to determine the function prior that yields the best performance results for a particular optimizer. Some initial results are discussed in the next sections.

Projection into a dual space is often used to prove theorems about the original space being studied. The nature of the performance dual is of interest because it can be used to analyze the theoretical performance of optimizer. The results such as those suggested in this subsection may shed further light on how optimizers perform under different performance criteria and function priors.

10.3 Linear Functionals

The duality mapping provides a source for generating linear functionals on the subspaces of $\mathcal{PF}$. In this section, some interesting consequences of duality and the linearity of performance are explored, including (1) the relationship of the performance dual to the continuous dual, (2) how performance can be improved automatically by following performance lines, and (3) how linear projections into Euclidean space can be used for similarity analysis.

10.3.1 Continuous Linear Functionals

For each non-constant random objective F, its prior $\mathbb{P}_F$ induces a linear functional over $\mathfrak{A}[\mathcal{MF}_{\text{tr}}]$ through the equation

$$\ell_F^\phi(\mathcal{G}) = \langle \mathcal{G}, \mathbb{P}_F \rangle_\phi \tag{10.6}$$

for a fixed performance criteria ϕ that is one of ϕ_w, ϕ_T, ζ_T, $\psi_\epsilon - 1$, $1 - \sigma_\epsilon$, or any other duality-inducing performance criterion. Denote the set of all such linear functionals for ϕ by $\mathcal{L}_\phi\left[\mathcal{NC}\right] = \{\ell_F^\phi : \mathbb{P}_F \in \mathcal{NC}\}$. These functionals will be called the *performance functionals* of a performance criterion ϕ.

Recall from Chapter 7 that the performance criterion is continuous if $\mathcal{G}$ is either continuous or sample convergent $\mathcal{G}_f$-*a.s.*. Therefore, ℓ_F is a continuous linear functional over subsets of $\mathfrak{A}[\mathcal{MF}_{\text{tr}}]$ containing only optimizers that are almost surely continuous under the joint distribution of F and Z. The standard (continuous) dual space of $\mathfrak{A}[\mathcal{MF}_{\text{tr}}]$ consists of all its continuous linear functionals, denoted as usual by $\mathfrak{A}^*[\mathcal{MF}_{\text{tr}}]$. The fact that at least some performance functionals may also be continuous suggests that there is some overlap between $\mathcal{L}_\phi$ and $\mathfrak{A}^*$.

In many cases, there is more than just some overlap. The space $\mathfrak{A}$ is a normed vector space, as is $\mathbb{R}$. A standard result of operator theory holds that any linear functional on $\mathfrak{A}$ is bounded if and only if it is continuous. Consider the standard *operator norm* for these functionals, given by

$$||\ell||_{\mathcal{L}} = \sup_{\{\mathcal{G}_f \in \mathfrak{A} : ||\mathcal{G}_f||_{\mathfrak{A}} = 1\}} \ell(\mathcal{G}_f). \tag{10.7}$$

That is, the operator norm of a linear functional on $\mathfrak{A}$ is the upper bound of that functional when applied to all long-running optimizers extended from the appropriate subset of $\mathcal{PF}$. For a given performance functional, $||\ell_F^\phi||_{\mathcal{L}}$ is bounded only if ϕ is bounded $\mathbb{P}_F$-*a.s.* over all of $\mathcal{G}_f$. Thus the space of functionals $\mathcal{L}_\phi$ consists entirely of bounded linear functionals whenever ϕ is one of $\psi_\epsilon^N - 1$ or $1 - \sigma_\epsilon$, and possibly in many other cases as well.

As a consequence, every performance functional for a bounded ϕ is also continuous, and $\mathcal{L}_\phi \subseteq \mathfrak{A}^*$. Given this observation, it seems reasonable to speculate that

$$\mathfrak{A}^* = \bigcup_{\phi \text{ bdd}} \mathcal{L}_\phi. \tag{10.8}$$

Importantly, the fact that $\ell_F^\phi = \mathbb{E}_{\mathbb{P}_F}\phi$ is continuous does not imply that ϕ is continuous over objectives. Rather, the continuity of ℓ_F^ϕ is an extension of the fact that ϕ is everywhere continuous over optimizers as proven in Theorem 7.3.5. It is also possible for a subset of $\mathcal{L}_\phi$ to be bounded as well. If F is almost surely bounded, then ℓ_F^ϕ is bounded when ϕ is one of ϕ_T, ζ_T, and sometimes ϕ_w. In these cases, ℓ_F^ϕ is also continuous over optimizers.

10.3.2 Performance-improving Linear Extensions

Because the performance functionals are linear, then performance on a particular random objective could be improved by extending the line between two optimizers. If F is a random objective and $\mathcal{G}, \mathcal{G}' \in \mathcal{O}_{\text{tr}}$ are optimizers with $\ell_F^\phi(\mathcal{G}) < \ell_F^\phi(\mathcal{G}')$ then supposing $\mathcal{G}'$ is not at the boundary of $\mathcal{O}_{\text{tr}}$, better performance under ϕ can be obtained by extending the line $\mathcal{A}_\alpha = \mathcal{G} + \alpha\,(\mathcal{G}' - \mathcal{G})$ to the boundary of $\mathcal{O}_{\text{tr}}$, recalling that $\mathcal{O}_{\text{tr}}$ is a closed convex set by Propositions 3.4.2 and 3.4.4. In this case, one seeks the largest $\beta > 1$ such that $\mathcal{A}_\alpha[t, f]$ remains a probability distribution $\mathcal{A}_\alpha$-*a.s.* for all $\alpha \leq \beta$. It is not immediately clear how to find the requisite β, or how to sample $\mathcal{A}_\alpha$ for $\alpha > 1$, but the possibility of optimizing optimizers in this way is an intriguing consequence of the formal theory. This line of thought is an interesting direction for future work.

10.3.3 Performance-based Linear Projections

The duality between optimizers and priors also suggests that it could be possible to decompose subsets of optimizers linearly onto a pseudo-basis induced by the choice of performance criterion. In this way, a set of optimizers can be projected into a lower dimensional space, where their similarity and relative nearness in terms of performance may be assessed. Section 8.5 and its associated figures illustrate a simple projection analysis of this type.

In general, the spaces $\mathfrak{A}$ and $\mathcal{M}_a$ are far too large to be characterized by a countable basis. Still, given any optimizer $\mathcal{G}_f \in \mathfrak{A}$ and a sequence of random objectives $F_1, \ldots, F_N$, the performance functionals $\ell^{\phi}_{F_1}, \ldots, \ell^{\phi}_{F_N}$ can be applied to $\mathcal{G}_f$ to project it into $\mathbb{R}^N$. This projection can be represented as

$$proj(\mathcal{G}, \phi, \{F_i\}_{i=1}^N) = \left(\ell^{\phi}_{F_i}(\mathcal{G})\right)_{i=1}^N, \tag{10.9}$$

where N may be infinite.

For a family of priors $\mathcal{F} \subseteq \mathcal{M}_a$ (countable or finite), $proj(\mathcal{G}, \phi, \mathcal{F})$ is a real vector of dimension $|\mathcal{F}|$. Because the functionals $\ell^{\phi}_{F_i}$ are not guaranteed to be orthogonal, the set formed from the projection of all optimizers in $\mathfrak{A}$ may be a manifold of lower dimension than $\mathbb{R}^{|\mathcal{F}|}$, and many optimizers will project to the same point, since $\mathbb{R}^{|\mathcal{F}|}$ is generally of lower dimension than $\mathfrak{A}$. The choice of performance criterion plays a key role in determining the capacity of the projection; a trivial performance criterion, for instance, projects all optimizers to a single point regardless of the random objectives used. Likewise, if the random objectives are similar, the projection they induce may be less powerful for distinguishing optimizers.

In Section 8.5, the random objectives forming the projection set were deterministic, consisting of the experimental benchmarks. The benchmark set covered a wide variety of function types, but this set was still small and non-orthogonal. It would be of interest to see whether a larger set of stochastic functions would result in similar relationships among the optimizers tested.

This section has advanced several interesting perspectives on how the performance functionals and other linear projections of optimizer performance can be used both to improve optimization and to study the performance of optimizers. A full development of this material is left as future work, discussed in

Chapter 14. But it is clear that the formal approach adopted in this dissertation makes it possible to articulate numerous theoretical and practical issues worthy of further study. For now, the discussion turns to a game-theoretic analysis of optimization that examines some of these issues in a slightly different setting.

10.4 The Optimization Game

The process of optimization can be treated as a two-player game pitting an optimizer against an objective function. The optimizer attempts to minimize the performance criterion; the objective function seeks to maximize it. This arrangement will be termed the Optimization Game, and it will be analyzed in this section. This point of view will lead to new insights about optimization. The existence of an NFL prior in every search domain will be proven. For non-NFL priors, the information-maximization principle will be introduced, which suggests how to identify the optimal optimizer for a fixed function prior. The information-maximization principle will then become the central theme in the remaining chapters of the dissertation.

10.4.1 Game Theory and Optimization

Game Theory was formally proposed by von Neumann and Morgenstern in *Theory of Games and Economic Behavior* [205]. It was intended as a formal framework within which the decisions of rational economic actors could be quantified and explained in terms of their available actions and the likely response to those actions by other actors.

Formally, a two-player zero-sum game consists of a set of strategies $\mathcal{X}$ available to the first player, a set of strategies $\mathcal{Y}$ available to the second player, and a value function $V : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ that maps a pair of strategies, one for each player, to the score that the first player obtains when those strategies are adopted by each player, respectively. The score for the second player is defined as $-V$, so that the two scores sum to zero. The zero-sum game is adversarial; one player wins, and the other loses. Each player seeks to maximize its value function.

This description from the prior paragraph is known as the *normal form* of a game. The strategies are termed *pure strategies* and are not allowed to involve random choices. In the Optimization Game, the first player optimizes an objective function, and the pure strategies it can play consist of the long-running trajectory-restricted deterministic optimizers $\mathfrak{A}\left[\mathcal{DF}_{\text{tr}}\right]$. The second player chooses an objective function to confound the optimizer, and its pure strategies are identified with the available objective functions, $\mathbb{R}^X$. The value function is given by the logarithm of a performance criterion, e.g. $V = -\log \zeta_{T_m}$, with the logarithm applied to yield positive and negative values and additively inverted so that maximizing the value function minimizes the performance criterion. The first player is referred to as the optimizing player, and the second player is referred to as the objective player.

In addition to deterministic, pure strategies, players may opt to play a *mixed strategy* if the game is repeated. In each repetition of the game, each player randomly selects a pure strategy according to some distribution and plays the selected strategy. The particular distribution over pure strategies is referred to as a mixed strategy. In the Optimization Game, the mixed strategies for the first player are the long-running trajectory-restricted optimizers, $\mathfrak{A}\left[\mathcal{O}_{\text{tr}}\right]$. The available mixed strategies for the second player are the the admissible function priors, $\mathcal{P}_a\left[\mathbb{R}^X\right]$. When mixed strategies are considered, the value function is

$$V\left(\mathcal{G}, \mathbb{P}_F\right) = -\log \left\langle \mathcal{G}, \mathbb{P}_F \right\rangle_{\zeta_{T_m}}.$$

In addition to the normal form, games have an equivalent representation as a series of iterated choices. In the Optimization Game, play proceeds in turns: The first player selects a point to be evaluated according to its chosen optimizer, and the second player evaluates that point according to its chosen objective. The history of evaluation points and their evaluations is fixed at each turn; neither player can undo its choices. In addition, the second player must be consistent with its prior evaluations. If it has previously evaluated a particular point, it must return the same evaluation as previously.

The mixed strategies used by the optimizing player in extensive form are the elements of $\mathcal{O}_{\text{tr}}$. Suppose the first player has adopted a strategy $\mathcal{G} \in \mathcal{O}_{\text{tr}}$. At time step $n+1$, the player samples $\mathcal{G}[Z_1^n, F(Z_1^n)]$ to choose an evaluation point $Z_{n+1} \in X$, using $Z_1^n = Z_1, \ldots, Z_n$ to represent the play history prior to time $n+1$ and $F(Z_1^n)$ to represent the history of observed function evaluations $F(Z_1), \ldots, F(Z_n)$. If the objective player is playing a strategy $\mathbb{P}_F \in \mathcal{P}_a\left[\mathbb{R}^X\right]$, then he responds by selecting an evaluation $F(Z_{n+1}) \in \mathbb{R}$ according to $\mathbb{P}_F\left(F(Z_{n+1}) \mid F(Z_1^n)\right)$.

The extensive form of a game is often represented as a tree, called the *game tree*. Each node of the tree represents a decision by one of the players, and the tree has one branch for each decision. In the Optimization Game, players may have infinitely many options. For this reason, many of the standard results of Game Theory, such as the Minimax Theorem, do not apply in general. However, if the search space is finite and the objective functions are restricted to take on finitely many values, the game tree representation is valid, and Minimax applies.

10.4.2 The Role of Information

The Optimization Game is a game of perfect information. At each time step, the players have access to the same information, consisting of the trajectory Z_1^n and its evaluations $F(Z_1^n)$. As described by von Neumann, the play history may be regarded as a filtration of σ-algebras, specifically, the filtration progressively generated by Z_1^n and $F(Z_1^n)$ [205]. This filtration gradually reveals the strategy of each player.

If the objective player is playing a mixed strategy $\mathbb{P}_F$, then the information contained in this filtration can be leveraged to produce a strategy for the optimizing player. As above, $Z_1^n = Z_1, \ldots, Z_n$ is the history of the first player's choices, and $F(Z_1^n) = F(Z_1), \ldots, F(Z_n)$ represents the second player's choices. Let $\mathcal{H}_n = \sigma\left(Z_1^n, F(Z_1^n)\right)$ be the σ-algebra generated by the histories Z_1^n and $F(Z_1^n)$. Then at a given time step n, consider

$$F_n(x) = \mathbb{E}\left[F(x) \mid \mathcal{H}_n\right]. \tag{10.10}$$

The function F_n is the conditional expectation of F with respect to $\mathcal{H}_n$. The conditional expectation is the closest random function to F out of all $\mathcal{H}_n$-measurable random variables using the L^2 norm over $\mathbb{P}_F$.[1] Equivalently, the

---

[1] This fact is a consequence of the Hilbert Projection Theorem, since $\mathbb{E}\left[G\left(F - F_n\right)\right] = \mathbb{E}\left[\mathbb{E}\left[GF \mid \mathcal{H}_n\right] - \mathbb{E}\left[GF_n \mid \mathcal{H}_n\right]\right] = \mathbb{E}\left[GF_n - GF_n\right] = 0$.

random error $F_n(x) - F(x)$ has lower variance than the error $G(x) - F(x)$ for any other random function G and any $x \in X$. That is, F_n is the best estimate of F given the information in $\mathcal{H}_n$.

10.4.3 The Objective Player Wins

When the function prior is such that evaluations reveal information about the objective function, then the best strategy for the optimizing player should leverage that information. In the Optimization Game, however, the objective player holds a trump card. He can select a strategy that intentionally hides information about the objective function so that evaluations effectively yield no information about the true minimum of the objective. Specifically, the objective player can win with an arbitrarily large score by playing a specially tailored NFL prior.

According to the Weak NFL Identification Theorem 9.3.8, an NFL prior is mean-constant and path independent. Weak NFL is used in this section because the value function defined above was based on the particular performance criterion, ζ_{T_m}. Path independence deprives the optimizing player of any information; under an NFL prior, the estimate $F_n(x)$ is a constant, since $F_n(x) = \mathbb{E}\left[F(x)\right]$. All optimizing strategies perform equally against an NFL prior. By adopting an NFL prior, the objective player fixes the score of the game to a constant, independent of the choices made by the optimizing player. In order to prove that the objective player wins with an arbitrarily large score, it needs to be shown that a suitable NFL prior exists, and that it can be constructed to yield an arbitrarily negative value for the game.

Theorem 10.4.1. *There exists an NFL prior for the Optimization Game that results in an arbitrarily large negative value for the game.*

Proof. This proof assumes that the space X is infinite. The result also holds for finite X if the m in ζ_{T_m} has $m < |X|$, but the finite case will not be handled here.

For the infinite case, the first step is to construct an NFL prior for an arbitrary space, which will be accomplished using the Kolmogorov's consistency theorems, as described in Section 6.1. First, recall that $\mathcal{B}\left[\mathbb{R}^X\right]$ is the σ-algebra generated by cylinder sets over the Borel σ-algebra $\mathcal{B}[\mathbb{R}]$, and $\mathcal{B}\left[X^{\mathbb{N}}\right]$ is the σ-algebra generated by cylinder sets over $\mathcal{B}_\tau$, the Borel σ-algebra on (X,τ). Define $\mathbb{P}_F$ to assign uniform probability to $F(x)$ on $[0,2M]$, for any $A \in \mathcal{B}[R]$ and any $x \in X$,

$$\mathbb{P}_F\left(F(x) \in A\right) = \frac{1}{2M}\int_0^{2M} 1_A(y)dy, \tag{10.11}$$

where 1_A is the indicator variable on the set A, i.e. $1_A(x) = 1$ if $x \in A$ and zero otherwise. Further, let separate values be independent of each other, so that for any $(A_1,\dots,A_n) \in \mathcal{B}[\mathbb{R}^n]$ and $x_1,\dots,x_n$,

$$\mathbb{P}\left(F(x_1) \in A_1,\dots,F(x_n) \in A_n\right) = \prod_{i=1}^{n} \mathbb{P}_F\left(F(x_i) \in A_i\right). \tag{10.12}$$

Equations 10.11 and 10.12 assign a probability to each cylinder set $B \in \mathcal{B}\left[\mathbb{R}^X\right]$. The finite-dimensional projections given by these equations are invariant under permutation, and the independence of the finite variables guarantees that the finite-dimensional distributions are consistent as the dimension increases. By Kolmogorov's consistency theorem, $\mathbb{P}_F$ has an extension to $\mathcal{B}\left[\mathbb{R}^X\right]$ whose finite-dimensional distributions match the two equations above.

Since all finite-dimensional projections are contained on the interval $[0,2M]$, their extensions are as well, and so $\mathbb{P}_F$ places probability one on bounded functions that reside in the interval $[0,2M]$ for all $x \in X$. From Equation 10.12, $\mathbb{P}_F$ is path independent. It is also mean-constant, since for all $x \in X$

$$\mathbb{E}\left[F(x)\right] = \int_0^{2M} y\,\mathbb{P}_F\left(F(x) \in dy\right) = \frac{1}{2M}\int_0^{2M} y\,dy = M.$$

Therefore $\mathbb{P}_F$ is weakly NFL on ζ_{T_m} by Theorem 9.3.8.

In order to complete the proof, the value of $\mathbb{E}F^*$ must be computed. Let $(x_n)_{n=1}^\infty$ be a countable sequence in X. Construct a countable subsequence $\left(x_{m_j}\right)_{j=1}^\infty$ by first letting $m_1 = 1$, and then for $j > 1$ choosing $m_j = k$ with $k > m_{j-1}$ so that $F\left(x_{m_j}\right) < 2^{-j}$. Such a countable subsequence cannot always be constructed, but the set of functions on which it can be constructed have probability one because

$$\mathbb{P}_F\left(F(x_n) \geq 2^{-j},\, \forall n > m_{j-1}\right) = \prod_{n=m_{j-1}+1}^{\infty} \mathbb{P}_F\left(F(x_n) \geq 2^{-j}\right) = \prod_{n=m_{j-1}+1}^{\infty} 1 - \frac{2^{-j}}{2M} = 0,$$

leveraging the mutual independence of $F(x_n)$. This fact is true for each j. Thus it is possible to choose $m_j < \infty$ with probability one. Because the sequence x_n was arbitrary, for all $\epsilon > 0$, $\mathbb{P}_F\left(F^* < \epsilon\right) = 1$ and $\mathbb{E}F^* = 0$.

$\mathbb{P}_F$ is weakly NFL on ζ_{T_m}, and thus $\mathbb{E}\left[\zeta_{T_m}(G, F)\right]$ is a constant for all $\mathcal{G} \in \mathcal{O}_{\text{tr}}$. Let $\mathcal{G}$ place probability one on some point $z \in X$. Then

$$\mathbb{E}_{\mathbb{P}_F}\left[\zeta_{T_m}(\mathcal{G}, F)\right] = \mathbb{E}_{\mathbb{P}_F}\left[F(z) - F^*\right] = M - 0 = M.$$

Thus the value of the game under $\mathbb{P}_F$ is $-\log M$ for the optimizing player. The choice of M was arbitrary and can thus be made arbitrarily large. $\square$

Theorem 10.4.1 proves that the objective player controls the game. Importantly, it also proves the existence in any search domain of an NFL prior that is not universally constant. However, the implications of this theorem should not be overwrought. An NFL prior corresponds to the philosophical position that the real world is of unbounded complexity and inherently unlearnable. This position is of little practical value. If nothing can be learned, no learning should be attempted. Yet the very experience of learning and predictability by humans and other animals nullifies the hypothesis that learning is impossible. The main lesson of Theorem 10.4.1 is that one is unlikely to encounter arbitrarily hard learning problems unless faced with a rational and adversarial intelligence.

Because learning is impossible under NFL, the remainder of this section assumes the objective player plays a fixed, non-NFL strategy. In this case, an optimizing strategy that seeks to maximize information may be optimal.

10.4.4 Optimal Optimization through Information-Maximization

In cases where the conditional expectation under the prior is computable, it is possible for the optimizing player to play a strategy that maximizes the information about the random objective. In fact, it is expected that such a strategy is optimal. This concept is formulated as the *information-maximization principle*

The Information-Maximization Principle. The optimal optimizer against a fixed function prior $\mathbb{P}_F$ is the one which fully utilizes the information obtained from prior evaluations in order to select new points optimally. Specifically, for a performance criterion $\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}[h(Z)]$ and the game history $\mathcal{H}_n$, define

$$g(x \mid \mathcal{H}_n) = \mathbb{E}_{\mathbb{P}_F}\left[h(Z) \mid \mathcal{H}_n, Z_{n+1}^{\infty} = x\right]. \tag{10.13}$$

with Z_{n+1}^{∞} representing the sequence $Z_{n+1}, Z_{n+2}, Z_{n+3}, \ldots$ and $x \in X^{\mathbb{N}}$. It is conjectured that an optimal strategy on a performance criterion ϕ chooses the point Z_{n+1} to be any member of the minimizing set for g:

$$O_{n+1} = \left\{y_1 \mid y \in X^{\mathbb{N}}, \forall x \in X^{\mathbb{N}}, g(y \mid \mathcal{H}_n) \leq g(x \mid \mathcal{H}_n)\right\} \tag{10.14}$$

for a given evaluation history $\mathcal{H}_n$. That is, the set of optimal moves at each time step consists of those moves that both improve the ultimate performance and simultaneously add the most information to a probabilistic model of the final performance.

The estimate of any random quantity with the least variance given a source of increasing information (i.e., a *filtration* as described in Chapter 6) is a *martingale*:

Definition 10.4.1. *Given a stochastic process $Z = (Z_n)_{n\in\mathcal{I}}$ for some strictly ordered index set $\mathcal{I}$ and a filtration $(\mathcal{F}_n)_{n\in\mathcal{I}}$ to which Z is adapted, Z is a martingale if for $m \leq n$,*

$$Z_m = \mathbb{E}\left[Z_n \mid \mathcal{F}_m\right].$$

Martingale theory is an important topic in the theory of stochastic processes, and the properties of martingales are generally well understood [38, 105]. In particular, a martingale can be generated by conditioning on a filtration. Such a martingale is known as a Levy martingale (or a Doob martingale); one example is the process $(\phi_n)_{n\in\mathbb{N}}$ defined by

$$\phi_n\left(\mathcal{G}, F\right) = \mathbb{E}\left[\phi\left(\mathcal{G}, F\right) \mid \mathcal{H}_n\right]. \tag{10.15}$$

The optimal optimizer as predicted by the information-maximization principle controls the filtration $\mathcal{H}_n$ in order to optimize the conditional expected performance ϕ_n. In essence, the proposed optimizer plans all future evaluation points to optimize its performance given the results of prior evaluations, and then proposes the first point of the optimal plan. A new plan is developed at each step to take the results of evaluation into account.

It is not entirely clear how to prove the information-maximization conjecture, and even if the conjecture is proven, it may not always be possible to find a point in the minimizing set O_{n+1} analytically, although it may be possible to approximate such a point. The next subsection proposes a strategy that attempts to implement the information-maximization principle heuristically.

10.4.5 Martingale Optimization

The information-maximization principle suggests a new approach to optimization that will be termed *martingale optimization.* A martingale optimizer generates an optimization process that is a martingale with respect to the filtration generated by the evaluation history. In this approach, the results of objective evaluation as an information source that the optimizer can use to choose which points to evaluate. The following paragraphs outline one possible martingale optimizer that can be used if the function prior is known.

The Optimization Game as defined above is scored with a value function based on the performance criterion ζ_{T_m}. Thus the optimizing player can spend $m - 1$ moves to develop a good model of the objective function around the optimum, and then one final move to guess the minimal point based on the model. Recalling F_n from Equation 10.10 and applying the information-maximization principle, the optimal final move is given by

$$Z_m = \text{argmin}_{x \in X} F_m(x), \tag{10.16}$$

which is the most rational estimate of the true minimum given the information in $\mathcal{H}_m$. The initial moves serve to prepare $\mathcal{H}_m$ so that it holds as much useful information as possible.

In order to maximize the information in $\mathcal{H}_m$ during the first $m - 1$ moves, the first player could attempt to minimize the variance of the estimate

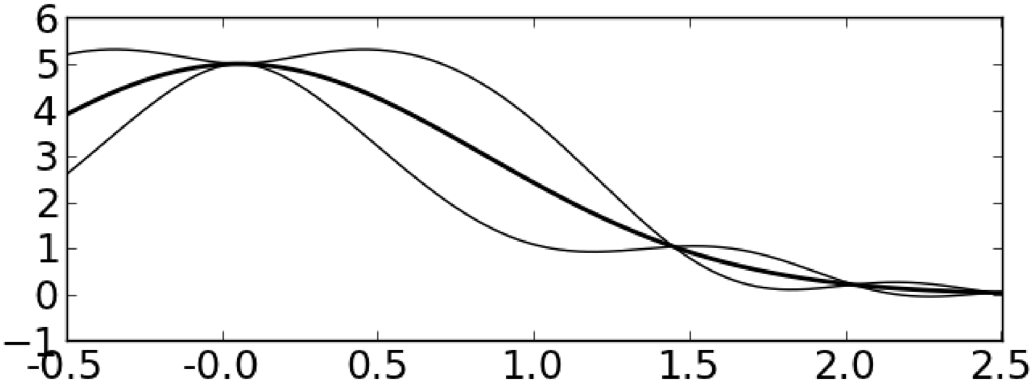


Figure 10.1: An example showing an estimated objective function such that the variance of the estimate increases with distance from the control points. The highest variance occurs at $x = 0.75$, but a new optimum is much more likely to be found near 2.25. When optimizing an objective function, merely reducing variance is not enough; the objective values must be taken into account as well. The optimal point to choose should trade off between reducing variance and improving the objective value.

F_m, since variance represents uncertainty, and the estimate is most likely to be mistaken in areas where its variance is high. However, merely minimizing the variance is not enough. In some places, the variance may be high, but the nearby values of the objective function are so large that the true minimum of the function is highly unlikely to reside in that region.

This situation is visualized in the context of a Gaussian process in Figure 10.1. In this case, it is more profitable to minimize the variance in regions where the objective value is low. Thus there is a tradeoff between removing uncertainty in general and removing uncertainty near the expected minimum. Proportional sampling provides one way to address this tradeoff. Let L_n be the lower variance-adjusted estimate of F,

$$L_n(x) = F_n(x) - \alpha\sqrt{\mathrm{Var}\left[F(x) \mid \mathcal{H}_n\right]}$$

for some α, with $\alpha = 1$ being the first standard deviation. Then, for $n < m$, the optimizing player can choose Z_n with probability

$$\mathbb{P}\left(Z_n \in dx\right) \propto \exp\left(\frac{-L_n(x)}{T}\right), \tag{10.17}$$

where T is a factor that controls the sensitivity of the strategy to different values of L_n. The probability in Equation 10.17 chooses points proportionally according to the variance-adjusted estimate L_n. It balances the choice of points near known good values against the need to increase evaluations in regions with high variance. This probability is similar to the Boltzmann distribution used by simulated annealing.

In summary, then, the proposed strategy for the optimizing player is to choose $m - 1$ points in succession according to Equation 10.17 followed by a single point chosen according to Equation 10.16. This strategy is an information-maximizing strategy. It attempts to develop a useful set of information near the apparent optima and then makes its best guess at the end. Such a strategy takes advantage of the control that the optimizing player has over which points can be evaluated. While the tradeoff between exploration and exploitation encoded in Equation 10.17 may not be optimal, it seems plausible that this strategy or a similar one could perform best against a given function prior $\mathbb{P}_F$ on the performance criterion ζ_{T_m}.

In order to implement this particular strategy, it must be possible to compute or approximate both $\mathbb{E}\left[F(x) \mid \mathcal{H}_n\right]$ and $\text{Var}\left[F(x) \mid \mathcal{H}_n\right]$. Thus this strategy is still not completely specified, and a computable function prior F is needed in order to instantiate it. In the next chapter, a simpler strategy named *evolutionary annealing* is proposed that ignores the conditional variance and assumes that the function prior is a random mixture sieve. This simpler strategy can be fully specified and efficiently implemented. Future studies will account for more complex priors as well as the conditional variance.

10.4.6 Curiosity Search

Curiosity Search, introduced by Schaul et al [177], partially implements the strategy described above. In Curiosity Search, the current set of points and its evaluations are used to construct a Gaussian process estimate of the objective function. The next evaluation point is selected by an internal optimization routine on the Gaussian process. If the objective function is drawn from a diffusion prior as described in Section 10.1.2, then the conditional expectation F_n is the mean value of the Gaussian process with a matching kernel. Thus Curiosity Search selects evaluation points according to Equation 10.16 above. To fully implement the information-maximizing strategy for ζ_{T_m}, Curiosity Search would need to be modified to intentionally minimize the variance. Without doing so, however, this method still obtains good results on optimization.

10.4.7 Strategic Forgetting

Genetic algorithms strategically forget all prior populations. As such they leak information. Given the discussion above, one might expect the strategy of forgetting to be a mistake that will always reduce performance. In actual practice, forgetting plays two roles. The first is practical. Particular algorithms can tend to focus evaluations on region of the search space where they have had success previously. This focusing behavior can create a feedback loop that pulls all future evaluations into a narrow region of the search space, resulting in convergence around a local minimum. Periodically forgetting old points can help preserve diversity among new evaluation points. However, an optimizer that uses information effectively need not be trapped by this feedback loop. An example of such an optimizer is evolutionary annealing, which is introduced in the next chapter. Also, notice that the most effective population-based algorithms on the static fitness functions in Chapter 8 have some means of retaining the most important information from past evaluations. DE keeps the best evaluation point along a sequence of parallel trajectories, and CMA-ES follows a gradient-based meta-strategy that tracks an evolutionary path.

The second benefit of forgetting is observed when the objective function is dynamic. The assumptions of this dissertation require the fitness function to be static and unchanging. Even in the context of function priors, the objective function, once evaluated, has been presumed to retain the same value on any repeated evaluation of a previously visited point. Allowing the objective function to be stochastic would not substantially change this analysis. Stochastic objective functions can be treated as a space of functions of the form $X \to \mathcal{P}[\mathbb{R}]$ instead of $X \to \mathbb{R}$, and many of the definitions and theorems would still be relevant. But in the case of a dynamic fitness function, there is no presumed regularity to repeated evaluations at the same point. A strong solution at one point in time could later become a poor solution due to nothing other than the passage of time. In a dynamically changing landscape, forgetting may be a viable strategy because it allows an optimizer to adapt to changing fitness conditions.

For these reasons, population-based optimizers that forget are most likely suboptimal on static or simply stochastic objective functions, but they can outperform other strategies on a dynamic objective. Dynamic objectives do occur in the real world, particularly in competitive domains, such as advertising, marketing, and games. In Chapter 8, optimizers were tested on static objectives, and population-Markov methods such as rGA, rBOA, and even SA fared poorly. Thus, particularly on a static objective, strategic forgetting is a poor strategy in comparison to strategies that preserve useful information over the entire evaluation history.

10.5 Conclusion

The NFL Identification Theorem showed that the NFL property implies path independence. Path independence in turn implies that learning and prediction are impossible. Given that learning and prediction are observed in reality, one must conclude that the reality prior is not subject to NFL. That is, general-purpose optimizers exist, and it makes sense to search for them. This concept was discussed from a pragmatic and philosophical point of view in Section 10.1, where it was conjectured that NFL produces hypothesis that necessarily violate Occam's razor. Thus, if smaller problem descriptions are more likely, then effective general-purpose black-box optimizers exist.

When the function prior is non-NFL, then performance varies over optimizers and priors. The duality from Section 10.2 provides a means of relating the performance of a class of optimizers to a range of function priors. This duality provides a source of linear functionals that can be used to project a set of optimizers into a finite Euclidean space, where their relationships with each other may be more easily analyzed.

In particular, in the case where the problem class is constrained by some fixed function prior, then one wishes to know which optimizer will perform best on the given problems. The Optimization Game formalizes this concept in game-theoretic terms. Since the game is dominated by NFL priors, which must always exist, the proper choice of optimizer is only relevant when the prior is fixed. In this case, a strategy that makes full use of the game state seems likely to perform best. A theoretical strategy implementing this idea was presented in Section 10.4.5 based on the information-maximization principle in Equations 10.13 and 10.14. In the next chapter, a concrete class of optimizers is proposed that implements aspects of this information-maximizing approach.

By this point, the potential power of the formalization adopted by this dissertation should be clear. This formal analysis brings mathematical tools to bear on traditional problems in optimization and permits new insights into what new types of optimizers might be discovered and how their performance may be assessed. In the final portion of this dissertation, these insights are applied concretely to propose evolutionary annealing, a practical information-based strategy that builds on the principles of performance from this chapter.

Chapter 11

The Evolutionary Annealing Method

In Chapter 10, an optimizer was conjectured to achieve its best performance on a non-NFL function prior by making full use of the information about the objective function obtained from function evaluations, and martingale optimizers were proposed as a consequence. In this chapter, *evolutionary annealing* is proposed as a practical martingale optimizer, i.e., a general-purpose optimization technique that efficiently uses past evaluations in order to select new evaluation points. Like simulated annealing, evolutionary annealing is a meta-technique that can be applied to many optimization tasks. This chapter introduces the basic algorithm and shows theoretically that instances of the algorithm converge to the global optimum under certain conditions. The next two chapters evaluate this approach experimentally in two specific spaces, finite-dimensional Euclidean space and neural networks.

11.1 Foundation

In the previous chapter, a martingale optimizer was defined as any optimizer whose optimization process is a martingale with respect to the evaluation history. Evolutionary annealing, introduced in this chapter, is a martingale optimizer. Much like the strategy described in Section 10.4.5, it chooses points proportionally to their expected objective value. It also adopts several simplifying assumptions that make it possible to implement the algorithm efficiently. The conditional variance is not used to select new points, and the particular form of the conditional expectation is determined by an implementation-specific family of mutation operators. Given the discussion of globally optimal optimizers in Chapter 10, it may seem disappointing that the practical algorithm proposed here does not implement many of the concepts suggested by the theory. However, it does apply the core ideas of martingale optimization, and the resulting optimizer is competitive with other state-of-the-art optimizers despite its limitations. The success of evolutionary annealing suggests that future work on developing efficient ways to implement more aspects of the information-maximization principle is likely to be rewarded.

11.1.1 Martingales vs. Markov Chains

Martingale optimization stands in contrast to optimization methods based on the convergence of Markov chains. Simulated annealing, for instance, converges globally in some instances because its sequence of accepted points generates an irreducible, aperiodic Markov chain that satisfies the principle of detailed balance [109]. Most evolutionary algorithms are also Markov; the population for each generation is constructed stochastically from only the population in the prior generation. As a result, these algorithms can discover and then forget high-quality regions within the search domain. They can therefore miss crucial information from the past, resulting in suboptimal performance.

This problem can be alleviated by selecting new evaluation points based on the entire pool of previously observed solutions. A genetic algorithm with non-Markovian selection can in principle become trapped in local optima by prematurely focusing on a narrow region of the search space. Evolutionary annealing combines genetic algorithms and simulated annealing using martingales in a manner that prevents this premature focus, resulting in an evolutionary algorithm that takes advantage of the full information gathered from the entire history of function evaluations. Evolutionary annealing solidly outperforms both genetic algorithms and simulated annealing, and compares favorably with the bank of stochastic optimization methods tested in Chapter 8.

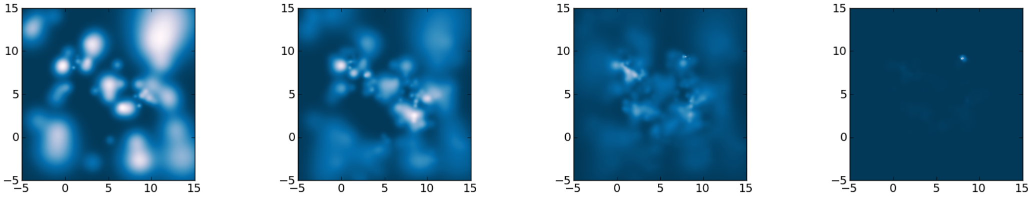
11.1.2 Characteristics of Evolutionary Annealing

Evolutionary annealing is a global optimization algorithm for Borel measure spaces that can be alternately viewed as a genetic algorithm with non-Markovian selection or as a method for performing simulated annealing without the Metropolis sampler. Evolutionary annealing introduces two annealed selection operators, exploiting a connection between the average effect of proportional selection and the annealed Boltzmann distributions used in simulated annealing. Although many genetic algorithms have previously employed the Boltzmann distribution for selection (e.g. [78, 102, 141]), evolutionary annealing is distinct from these approaches in that it can select any member of any prior population and does so using information generated by a sequence of refining partitions of the search domain. Evolutionary annealing is distantly related to Estimation of Distribution Algorithms (EDAs), since it builds a global model of the annealing distributions for the fitness function (see Section 2.7.2, [140, 154]). However, whereas EDAs build models based solely on the best members of the immediately prior generation, evolutionary annealing maintains a martingale model based on the entire history of observation. By leveraging the information acquired from function evaluations, evolutionary annealing builds an increasingly refined estimate of the fitness function that allows it to locate the global optimum. To illustrate this process, the progress of an example run of evolutionary annealing in a two-dimensional space is shown in Figure 11.1.

Theoretically, evolutionary annealing converges asymptotically to the true global optima of the fitness function. The proof is given in Section 11.3.1. Experimentally, evolutionary annealing converges at a controlled rate as demonstrated on the twelve global optimization benchmarks from Chapter 8. Because of its efficient use of information gained from evaluations, evolutionary annealing performs well in a comparison with the other optimization methods evaluated in Chapter 8, i.e. simulated annealing (SA), differential evolution (DE), evolution strategies with correlated matrix adaption (CMA-ES), particle swarm optimization (PSO), the real-coded Bayesian optimization algorithm (rBOA), a real-coded genetic algorithm (rGA), the Nelder-Mead algorithm (NM), a basic generating set search (GSS), and conjugate gradient descent (CG).

11.1.3 Expected Proportional Selection

Evolutionary annealing builds on concepts from simulated annealing and evolutionary algorithms (discussed in Section 2.5.1, Section 2.6, and Chapter 4). There is an interesting theoretical connection between genetic algorithms and simulated annealing that motivates the global selection mechanism of evolutionary annealing. This connection is exposed by trivial manipulations



(a) 50 points (b) 125 points (c) 250 points (d) 500 points

Figure 11.1: Example run of evolutionary annealing on Shekel's Foxholes in two dimensions (shown in Figure 8.1(e)). Images are heat maps displaying the estimated probability density of evolutionary annealing, that is, the probability that each point will occur in the next generation of evolutionary annealing. White areas are more probable, and dark areas are less probable. Successive frames show how the probability density changes once 50, 125, 250, and 500 points have been evaluated. The resulting distribution increasingly models the fitness function; comparison with Figure 8.1(e) confirms that after 500 evaluations, evolutionary annealing has focused on the true global optimum.

of a previous result of Mühlenbein and Mahnig [141], as will be discussed in this subsection.

Many genetic algorithms employ *proportional selection*, where individuals in the prior population are selected proportionally to their observed fitness (see Section 4.2.3). Much like simulated annealing, proportional selection sharpens the fitness function implicitly with each generation, so that on averaging over population trajectories the selection operator asymptotically places probability one on the optima of the fitness function. Following Mühlenbein and Mahnig [141], proportional selection at the n^{th} time step is given by $\mathcal{S}_f^n(x) \propto f(x) N_x^{n-1}$, where $\mathcal{S}_f^n(x)$ is the probability of selecting x at time n, and N_x^n is a random variable indicating the number of copies of the solution x in the population at time n. Taking the expected value over N_x^n,

$$\mathbb{E}\left[\mathcal{S}_f^n(x)\right] \propto f(x)\mathbb{E}\left[N_x^{n-1}\right]. \tag{11.1}$$

The expected value on the left is also a probability distribution over x, here termed *expected proportional selection*. It differs from proportional selection in that expected proportional selection may assign positive probability to any point in the search domain. It is possible to imagine an evolutionary algorithm where each successive population is sampled from just this rule. This algorithm is a one-stage, selection-only genetic algorithm; because expected proportional selection averages over all individuals, no variation is required.

In such an algorithm, if the initial population is selected uniformly at random, then $\mathbb{E}\left[N_x^0\right]$ is a constant, so

$$\mathbb{E}\left[\mathcal{S}_f^1(x)\right] \propto f(x). \tag{11.2}$$

By definition, $\mathbb{E}[\mathcal{S}_f^n(x)] = \mathbb{E}[N_x^n]/K$ where K is the population size, since N_x^n/K is just the proportion of the population taking the value x. Applying this fact to the recursion in Equation 11.1 yields $\mathbb{E}[S_f^n(x)] \propto f(x)^n$. Thus expected proportional selection sharpens the fitness function. Introducing $g(x) \equiv -\log\left(f(x)\right)$,

$$\begin{aligned}\mathbb{E}\left[S_f^n(x)\right] &\propto \exp\left(-g(x)\right)^n \\ &= \exp\left(-\frac{1}{n^{-1}}g(x)\right). \qquad (11.3)\end{aligned}$$

Comparing Equation 2.3 to Equation 11.3, expected proportional selection is found to have an annealing distribution on $-\log f$ with cooling schedule $T_n = n^{-1}$. Since the logarithm is monotonic, the maxima of f are the minima of g.

Expected proportional selection is not a feasible selection rule, because it requires total knowledge of the fitness function a priori. If such knowledge were possible, there would be no need for iterative optimization. The optima would already be known. Expected proportional selection could be estimated by averaging over the trajectories of several different runs of a genetic algorithm, but the number of trajectories required for a good estimate would be intractably large. Genetic algorithms with proportional selection can be viewed as an approximation of expected proportional selection.

Evolutionary annealing exploits the theoretical relationship between simulated annealing and genetic algorithms to create a hybridized algorithm that merges qualities of both algorithms, as will be described next.

11.2 The Evolutionary Annealing Approach

This section defines the evolutionary annealing algorithm. The formal context and notation are introduced first, followed by the algorithmic details.

11.2.1 Formal Context and Assumptions

Reviewing and expanding the notation from Section 3.2.3, let the search domain (X, τ) be a topological space with a given Hausdorff (separated) topology, and let $(X, \mathcal{B}_\tau)$ be a measurable space such that $\mathcal{B}_\tau$ is the Borel σ-algebra for the given topology on X. By this formulation, open sets are always $\mathcal{B}_\tau$-measurable. Evolutionary annealing is defined with respect to a base measure λ that is finite on $(X, \mathcal{B}_\tau)$ and positive on all open sets. Let $f : X \to \mathbb{R}$ be a fitness function which is to be minimized, and assume that f has all necessary integrability properties required by the formulae that follow. Primarily, $\exp(-f/T)$ must be integrable for bounded $T > 0$. The notation (P_n) will represent a *stochastic population process*, that is, a sequence of populations generated by a stochastic optimization algorithm. Each population P_n contains a fixed number of individuals and is denoted by $P_n = \left(P_n^k\right)_{k=1}^K$, where K is the population size. For a given trajectory $t \in \mathcal{T}[X]$ with $|t| = NK$, this definition implies that $P_n^k = H(t)^{n,k}$ for $n \leq N$. The set A_n represents the set of all individuals up to time n, $A_n = \bigcup_{m \leq n,k} \left\{P_m^k\right\}$. With these definitions, the basic algorithm can be defined.

11.2.2 Basic Algorithm

Evolutionary annealing consists of selection and variation phases. The population P_{n+1} is sampled one individual at a time in these two stages. In the selection phase, an element $a \in A_n$ is selected with probability

$$p_n(a) = \xi_n \exp\left(-\frac{f(a)}{T_n}\right) \lambda\left(E_n^a\right), \tag{11.4}$$

where T_n is a cooling schedule, ξ_n is a normalizing factor, and $\lambda\left(E_n^a\right)$ is the measure of a region surrounding the point a, discussed below. This selection mechanism will be termed *annealed proportional selection* based on the relationship between expected proportional selection and annealing described in the prior section. Using the formalisms introduced in Chapters 3 and 4, annealed proportional selection may be written as

$$\mathcal{APS}\left\langle T\right\rangle[t,f](\{a\}) = 1_{a\in H(t)}p_{|H(t)|}(a), \tag{11.5}$$

where $T = (T_n)_{n=1}^{\infty}$ is the cooling schedule and $1_{a\in H(t)}$ is used to ensure that the set $A_{|H(t)|}$ has probability one, as required by the formal definition of a selection rule. The primary distinction of $\mathcal{APS}$ is that it can select any member of any prior population.

For the variation phase, evolutionary annealing requires a family of probability distributions $\{\nu_n^x\}_{x\in X}$ used to mutate selected points, so that given a selected point x, ν_n^x is used to vary x at time n. The choice of mutation distributions is essentially arbitrary, although the convergence theorems that follow will restrict this choice. In Euclidean space, Gaussians can be used, centered at x and with the variation as a hyperparameter σ_n. In binary spaces, individual bits can be flipped with a probability dependent on n. The particular mutation distributions should be chosen based on the needs of the problem at hand; a mutation distribution whose shape is well matched with the objective function will converge much faster than one that is not. The choice of mutation distribution determines the function prior with respect to which evolutionary annealing is best aligned in the sense of Section 10.2. Some results for a specific instantiation of evolutionary annealing with real vectors will be discussed in Section 12.2. The family of mutation distributions defines a mutation operator in the terminology of Chapter 4 through the equation

$$\mathcal{V}[t\cup x, f](A) = \nu_{|H(t)|}^{x}(A). \tag{11.6}$$

Once an individual $a \in A_n$ has been selected with probability $p_n\left(a\right)$, then that individual is mutated according to ν_n^a in order to generate a new member of the population. That is, each individual in the population at time $n+1$ is sampled according to

$$P_n^k \sim \sum_{a\in A_n} p_n(a)\nu_n^a\left(dx\right). \tag{11.7}$$

Thus evolutionary annealing samples its populations from a sequence of mixture distributions with one mixing point located at each individual from prior populations. In this way, the selection is non-Markovian; the selected individual could come from any previous generation. The mixture probabilities $p_n(a)$ are chosen according to the annealing formula in Equation 11.4.

Equation 11.7 may be recognized as a convolution, and so evolutionary annealing with annealed proportional selection may be written as

$$\mathcal{EA}\left\langle T, \mathcal{V}\right\rangle = \mathcal{APS}\left\langle T\right\rangle \star \mathcal{V}, \tag{11.8}$$

reflecting a dependence on the cooling schedule and the choice of mutation distributions.

Intuitively, if the temperature is fixed at a constant, as the number of mixing points increases and the variance of the mutation distribution decreases, the mixture distribution in Equation 11.7 converges to the annealing distribution $\mathcal{A}_n^f$ in Equation 2.3. It is commonly known that mixtures of Gaussians can model any sufficiently smooth distribution arbitrarily well if enough mixing points are used. It is also true that mixture distributions in general can model any probability measure arbitrarily well subject to certain conditions. A specific proof of convergence for evolutionary annealing is offered in Section 11.3.1; Theorem 11.3.1 states that evolutionary annealing converges in probability to the optima of f. Therefore P_n is successively sampled from better and better approximations to $\mathcal{A}_n^f$, and as $n \to \infty$, the population sequence P_n will increasingly focus on the optima of f. The rate of convergence will be taken up in Section 11.3.2.

A high-level algorithm for evolutionary annealing over N generations is shown in Algorithm 1. The algorithm depends on two subroutines, *prepare* and *sample*. The subroutine *prepare* builds data structures to support efficient sampling of the quantity p_n from Equation 11.4. The subroutine *sample* samples from p_n using the prepared data structures. Through the use of highly precise approximations as described in Section 12.1.2, both *prepare* and *sample* can be implemented to run in time logarithmic in the population size and the number of generations. The specific implementations of *prepare* and *sample* used in the experiments utilize the methods of Section 12.1.2. The *prepare* routine adds nodes to the trees described in that section and propagates the components of Equations 12.1 and 12.7 up the tree. The *sample* routine employs Equations 12.1 and 12.7 to traverse the tree down from the

Algorithm 1 Evolutionary Annealing Algorithm

```
  $N$, the number of generations
  $K$, sample points (population size) per generation
  $\left(P_1^k\right)_{k=1}^K$, the initial random population
  $A_0 \leftarrow \emptyset$, all points from all generations
  for $n \leftarrow 1$ to $N$ do
    $A_n \leftarrow \bigcup_k P_n^k \cup A_{n-1}$
    $p_n \leftarrow prepare\left(A_n\right)$
    for $k \leftarrow 1$ to $K$ do
      $y \leftarrow sample\left(p_n\right)$
      $P_{n+1}^k \leftarrow$ a sample from $\nu_n^y$
    end for
  end for
```

root in order to select a previously evaluated point. Assuming that sampling ν_n^a and computing $\lambda\left(E_n^a\right)$ do not add to the complexity, the overall algorithm has performance $O\left(NK \log NK\right)$.

In order to make evolutionary annealing concrete, the cooling schedule must be determined. In light of [82], a default choice for the cooling schedule is given by $T_n^{-1} = \eta \log n$. Here η is a learning rate that scales the fitness function and thereby controls the aggressiveness of selection. A high learning rate focuses selection on the few best individuals and may restrict exploration of the space. A low learning rate allows promiscuous selection, slowing down refinement of previously discovered solutions but increasing the probability of escaping a local minimum. Again following [82], a possible value for η is $1/d$ where d is the largest depth of a local minima relative to its surroundings in the fitness landscape. In more complex spaces, different cooling schedules could be considered. There may also be a benefit to linking the variance of the mutation distribution to the cooling schedule, so that as the probability of selecting the current best individual decreases, the variance also decreases to enable refined exploration of the immediate region around the current best. The effect of parameter settings is explored further in Section 11.3.2.

The region weight $\lambda\left(E_n^a\right)$ is present in Equation 11.4 to avoid a particular scenario of premature convergence. Once a good solution is discovered, evolutionary annealing will devote increasing resources to exploring the neighborhood of that point. If these points are also good, then the probability of selecting more points in the same region will increase in a feedback loop. Within a few generations, almost all points selected will come from the immediate environment of these good points. If there is a local minimum in the vicinity, evolutionary annealing would likely become entrapped in that region. The region weight is intended to serve as a measure of how many individuals have been previously sampled in the region surrounding the point a. The sets E_n^a partition X around points in A_n, the total population so far. Such a partition can be computed in logarithmic time in many spaces. These partitions also play an important role in the convergence proof in Section 11.3.1.

11.2.3 Partitioning the Space

To demonstrate convergence of evolutionary annealing, each of the mixing points $a \in A_n$ will be considered representative of a particular region of the search space X. Each successive set A_n will be associated with a partition $\{E_n^a\}_{a\in A_n}$ of disjoint sets such that $X = \bigcup_{a\in A_n} E_n^a$ and $a \in E_n^a$ for all n. The σ-algebra $\mathcal{F}$ is assumed to be rich enough to support such partitions based on any finite collection of points in X. The partitioning set E_n^a is the same as the one that appears in Equation 11.4.

Provided that there exists a computable algorithm to split any set containing two distinct points into two disjoint sets each of which contains exactly one of the points, then the partitions can be stored in a binary tree, and if the splitting algorithm does not depend on the population size of the number of generations, the computational complexity of maintaining a partitioning tree is logarithmic on average.

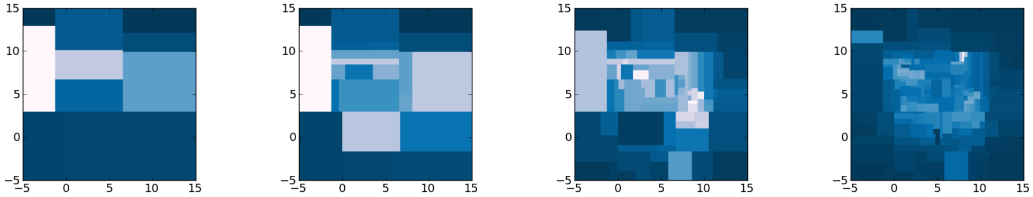
Algorithm 2 partitions any Borel measure space over a Hausdorff topology given a function for dividing a partition region between two separate points in the region. A partition is represented as a binary tree, with the root representing the entire space X and each branch partitioning X into two sets. The algorithm is initialized with a sequence of points $\{x_m\}_{m=0}^{M} \subseteq X$ to be partitioned (the mixing points), a tree $\mathcal{T}$ with X as the root node, and an assignment function k such that $k(m)$ is the leaf node of the tree assigned to the point x_m, or $\emptyset$ if no assignment has been made. The algorithm then

Algorithm 2 Algorithm to Generate a Partition Based On Grid Points

```
$\{x_m\}_{m=1}^{M} \subseteq X$, the mixing points
$\mathcal{T} \leftarrow \{X\}$, the partition tree
$k(i) \leftarrow \emptyset$ for all $i = 1, \ldots, M$, node assignment function
for $m \leftarrow 1$ to $M$ do
    $N \leftarrow$ the leaf node in $\mathcal{T}$ such that $x_m \in N$
    if $\exists j \neq m$ s.t. $k(j) = N$ then
        $N_0, N_1 \leftarrow$ separate $(x_j, x_m, N)$
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{N_0, N_1\}$
        $k(j) \leftarrow N_0$, $k(m) \leftarrow N_1$
    else
        $k(m) \leftarrow N$
    end if
end for
```

loops through the mixing points, splitting the space where necessary to ensure that each leaf node contains exactly one mixing point. The algorithm relies on *separate*, a domain-specific subroutine to split an existing set. At the end of each iteration of the algorithm's main loop, each leaf node is assigned to exactly one mixing point. When a new mixing point is added, *separate* partitions the leaf node to which it belongs into two new leaf nodes, each containing only one mixing point. The process of adding a single new mixing point to the tree requires only a tree traversal, so that at each generation, updating the partition requires $O(K \log NK)$ time, where NK is the number of points at the N^{th} generation.

In a vector space, such as $\mathbb{R}^d$, the function *separate* can in many cases be given explicitly. Suppose that X is bounded above by $\{u_i\}$ and below by $\{\ell_i\}$ so that X has a rectangular shape. Each node in the partition tree will restrict the coefficient for exactly one of the basis vectors, say j. To maintain computability, it is necessary to require that $j < D < \infty$ for some D. That is, each set E_n^a in the partition is defined as a hyperrectangle on finitely many coordinates, with each step in the traversal of the partitioning tree adding a new coordinate value for some side of the hyperrectangle. So E_n^a can be represented as two vectors, u^a for the upper bounds, and ℓ^a for the lower bounds. Given the point $a \in X$ and a second point $x \in X$, E_n^a can be separated as follows.



(a) 10 points (b) 25 points (c) 100 points (d) 250 points

Figure 11.2: Progression of partition regions $\{E_n^a\}$ on Shekel's Foxholes during the run of evolutionary annealing in Figure 11.1 in two dimensions. Images are heat maps displaying the selection probability of each region; light regions have a higher selection probability. Successive frames show how the partition regions gradually model the shape of the fitness function after 10, 25, 100, and 250 points have been evaluated.

Let $k = \text{argmax}_{i \leq D} |a_i - x_i|$; k is the index at which the rectangle E_n^a will be split. Suppose $a_k > x_k$ for the sake of simplicity; the opposite situation is handled analogously. Initialize $u^x \leftarrow u^a$ and $\ell^x \leftarrow \ell^a$. Then set $\ell_k^a \leftarrow \frac{1}{2}(a_k + x_k)$ and $u_k^x \leftarrow \frac{1}{2}(a_k + x_k)$. The regions E_{n+1}^a and E_{n+1}^x defined by these boundary vectors are then disjoint if the upper boundary is strict. The result of this partitioning method in $\mathbb{R}^2$ is shown in Figure 11.2. This version of *separate* cannot separate two vectors that are the same in the first D coefficients. In an infinite-dimensional vector space, it is possible for two distinct vectors to have arbitrarily many identical coefficients, and no computable algorithm can locate the coefficients in which they differ. This situation is of theoretical more than practical concern, however, and can be ignored in most cases. The separation algorithm above can be implemented efficiently in many spaces of interest. Section 12.1.2 discusses how these partition mechanisms can be used to implement the subroutines *prepare* and *sample* from Algorithm 1.

11.2.4 Annealed Tournament Selection

Annealed proportional selection as given in Equation 11.4 is a proportional selection rule; individuals are selected according to their proportion of the overall fitness. Proportional selection has a well-known drawback that also applies to annealed proportional selection. For example, suppose that the fitness function f has a minimal value of 0, and consider the selection probabilities for the points x, y with $f(x) = 0.01$ and $f(y) = 0.001$ at temperature $T_n = 5$. Assume $\lambda(E_n^x) = \lambda(E_n^y) = 1$. Then $p_n(y)/p_n(x) = 1.0018$. That is, x is almost equally as likely to be selected as y, even though y is a whole order of magnitude closer to the optimum. Thus the more precise solution is no more likely to be selected than rougher solutions close to the optimum, which makes refinement of solutions near a local or global optimum sluggish. These intuitions are confirmed by the experimental results in Chapter 12; annealed proportional selection converges within 0.1 of the optimal fitness without difficulty, but then fails to attain accuracy within 0.001 in most cases.

To address this weakness of proportional selection in genetic algorithms, tournament and ranking selection were introduced (cf. [201]). These methods select among individuals according to their fitness rank in the population rather than according to their raw fitness. For tournament selection, the best individual is selected with some probability q, termed the *selection pressure*. If the best individual is not selected, then the second best individual is chosen with probability q. Thus the probability of selecting the n^{th}-ranked individual of the population is proportional to $q(1-q)^{n-1}$.

A similar concept can be used to define *annealed tournament selection*, a non-Markovian version of tournament selection. Annealed tournament selection replaces Equation 11.4 by

$$p_n(a) = \xi_n \; q^{1/T_n} \left(1 - q^{1/T_n}\right)^{r(a)} \; \lambda(E_n^a), \tag{11.9}$$

where q is the selection pressure, and $r(a)$ is the fitness rank of a in A_n starting with 0. Annealed tournament selection uses a cooling schedule T_n so that the rank becomes increasingly significant with each generation, with the ultimate result that the top-ranked individual is selected at zero temperature. The main difference from standard tournament selection is that each individual must be ranked against all other individuals from all prior generations. As a consequence, the selection pressure must be much lower. For this paper, the value of q was fixed at 0.025. Rather than varying q, the learning rate η in the cooling schedule can be varied to achieve the same effect.

In terms of the formalism advanced in this dissertation, annealed tournament selection is given by

$$\mathcal{A}\mathfrak{T}\mathfrak{S}\left\langle T\right\rangle[t,f](\{a\}) = 1_{a\in H(t)}p_{|H(t)|}(a), \tag{11.10}$$

where p_n comes from Equation 11.9. Evolutionary annealing with annealed tournament selection may be defined as

$$\mathcal{E}\mathcal{A}_{\text{tour}}\left\langle T,\mathcal{V}\right\rangle = \mathcal{A}\mathfrak{T}\mathfrak{S}\left\langle T\right\rangle \star \mathcal{V}. \tag{11.11}$$

With the basic algorithm defined, it can now be shown that evolutionary annealing converges to the global optimum with either type of annealed selection. These results will be confirmed experimentally in Chapter 8.

11.3 Convergence Properties

Subject to a reasonable set of conditions, evolutionary annealing with either proportional or tournament selection converges in probability to the set of optimal points for the fitness function. These conditions include: (1) the mutation variance must asymptotically decrease faster than the partition size; (2) the annealing distributions must possess quasi-differentiability properties at the mixing points; (3) the fitness function must not be too irregular in the immediate neighborhood of the global optima; and (4) the mutation variance and the temperature must decay slowly enough to guarantee full exploration of the space. With these conditions satisfied, evolutionary annealing converges to the global optima. The convergence rate for evolutionary annealing is highly sensitive to both the cooling schedule and the variance decay; the interaction of these parameters remains the subject of inquiry.

In this section, the preceding concepts are made rigorous, and a proof of convergence for evolutionary annealing is provided, followed by a discussion of convergence rates. Several symbols are used, and a table of their meanings is provided in Table 11.3 to aid the reader.

11.3.1 Convergence Proof

As mentioned above, the convergence proof requires conditions on the mutation variance, the annealing distributions, and the fitness function. In-

Table 11.1: Table of symbols relating to the convergence proof

Symbol	Meaning
X	The search domain, a Hausdorff (separated) topological space
$\mathcal{B}_\tau$	The Borel σ-algebra on X
λ	A measure on $(X, \mathcal{B}_\tau)$ that is positive on open sets
f	A λ-integrable fitness function with finite minimum
f^*	The minimum of the fitness function
X_ϵ	The set of ϵ-optimal points in X, $\{x : \lvert f(x) - f^*\rvert < \epsilon\}$
A_n	The set of observed individuals at generation n
A	The limiting set of observed individuals as $n \to \infty$
T_n	A cooling schedule, $T_n \downarrow 0$
E_n^a	A set containing $a \in A_n$, with $\{E_n^a\}_{a \in A_n}$ partitioning X
$\lambda(E_n^a)$	The volume of the partition set E_n^a under the measure λ
ξ_n	A normalizing factor for $p_n(a)$
$p_n(a)$	The selection probability for $a \in A_n$, $p_n(a) = \xi_n \exp(\frac{-f(x)}{T_n})\lambda(E_n^a)$
ν_n^a	The mutation distribution around a point $a \in A_n$ at generation n
$\mathbb{G}_n$	The distribution of evolutionary annealing at generation n
g_n	The annealing density for the fitness function at temperature T_n
$\mathcal{A}_n$	The annealing distribution for the fitness function at temperature T_n
$\mathcal{A}$	The limit of the annealing distributions under the total variation norm
g_n^λ	The neighborhood average of g_n on E_n^a
λ_n^a	A measure on E_n^a given by $\lambda_n^a(B) = \lambda(B \cap E_n^a)/\lambda(E_n^a)$

tuitively, evolutionary annealing converges because it approximately samples from the annealing distributions with respect to the measure λ. Specifically, define

$$g_n(x) = \frac{\exp\left(-f(x)/T_n\right)}{\int_X \exp\left(-f(x)/T_n\right)\,\lambda\left(dx\right)}. \tag{11.12}$$

and note that g_n is the density of an annealing distribution generalized to the space $(X, \mathcal{F}, \lambda)$, i.e. $\int_X g_n \, d\lambda = 1$. Define the annealing distributions by

$$\mathcal{A}_n(B) = \int_B g_n(x)\lambda(dx) \tag{11.13}$$

so that $\mathcal{A}_n(X) = 1$ and consider the limiting distribution $\mathcal{A} \equiv \lim_n \mathcal{A}_n$, meaning $\mathcal{A}_n(B) \to \mathcal{A}(B)$ for all $B \in \mathcal{F}$. The functions g_n are positive, and therefore $\mathcal{A}_n$ is a probability measure. The definition of $\mathcal{A}$ implies $\mathcal{A}$ is positive and $\mathcal{A}(X) = 1$, so $\mathcal{A}$ is a probability measure as well. Specifically, $\mathcal{A}$ assigns measure zero to all non-optimal points of f.

In order to guarantee that the mixture distributions used by evolutionary annealing are capable of approximating g_n, it is necessary that the densities g_n do not vary too quickly, i.e., that the fitness function does not oscillate wildly between infinitesimally close points. Formally, this concept can be defined based on the integrals of g_n on *nicely shrinking* sets. Nicely shrinking sets are a vanishing sequence of sets, each of which possesses some interior volume. The following definition suffices for the purposes of this dissertation.

Definition 11.3.1. *A sequence of $\mathcal{B}_\tau$-measurable sets $\{E^a_n\}_{n\in\mathbb{N}}$ shrinks nicely around a point a if (1) for all n there is an open set I_n with $a \in I_n$ and $I_n \subseteq E^a_n$ and a constant $\alpha > 1$ such that $\lambda(E^a_n) < \alpha\lambda(I_n)$, and (2) for any open set O containing a, there is an $N < \infty$ such that $\mathcal{E}^a_n \subseteq O$ for all $n > N$.*

Definition 11.3.2. *Given a sequence of sets $\{E^a_n\}_{n\in\mathbb{N}}$ that shrink nicely around a point a and a sequence of functions $\{g_n\}_{n\in\mathbb{N}}$ on a measure space $(X, \mathcal{F}, \lambda)$ such that each g_n is λ-integrable, the neighborhood average of g_n on E^a_n is given by*

$$g^\lambda_n\left(a\right) \equiv \lambda\left(E^a_n\right)^{-1} \int_{E^a_n} g_n d\lambda. \tag{11.14}$$

Definition 11.3.3. *On a measure space $(X, \mathcal{F}, \lambda)$, a sequence of λ-integrable functions g_n is approximated by its neighborhood average at a point a if for any sequence of nicely shrinking sets $\{E_n^a\}$*

$$\left|g_n\left(a\right) - g_n^{\lambda}\left(a\right)\right| \to 0. \tag{11.15}$$

If the neighborhood average g_n^{λ} of a sequence g_n approximates the values of the sequence at a point well, then the neighborhood average can be used as a proxy for the function at that point. Approximation by the neighborhood average is a critical requirement for the convergence of evolutionary annealing, but is not overly restrictive in practical terms. This property is possessed by all continuous functions, but it is true for many discontinuous functions as well. In fact, only fitness functions that are chaotic at an infinitesimal scale are excluded by this requirement.

The next set of conditions pertains to the ϵ-optimal sets of f. Let f^* be the minimal value of f, and define $X_\epsilon \equiv \{x : f(x) < f^* + \epsilon\}$. X_ϵ includes all points in X that come within ϵ of the optimum. If the set X_ϵ has λ-measure zero for small values of ϵ, then the optima are isolated, and the mutation distributions for evolutionary annealing have zero probability of proposing the optima. In that situation, convergence is impossible.

A second pathological situation occurs when the boundary of the set X_ϵ is so jagged that it possesses positive λ-mass. In this case, the boundaries of X_ϵ can never be well approximated by a countable sequence of estimates.

A fitness function will be called *suitable* when these cases can be excluded. Additionally, suitability will be defined to account for the required integrability and neighborhood properties discussed above.

Definition 11.3.4. *A fitness function f is termed suitable on a particular cooling schedule $(T_n)_{n\in\mathbb{N}}$ whenever the following five conditions hold:*

1. *The minimum exists, i.e. $f^* > -\infty$.*
2. *The functions g_n are λ-integrable.*
3. *The functions g_n are well approximated by their neighborhood average.*

4. *The sets* X_ϵ *are* $\mathcal{B}_\tau$*-measurable,* $\epsilon \geq 0$.

5. *There exists a constant* $\gamma > 0$ *such that for all* $\epsilon \in (0, \gamma)$, $\lambda(X_\epsilon) > 0$ *and* $\lambda(\partial X_\epsilon) = 0$.

In the convergence proof that follows, the mutation distributions must be well matched with the base measure λ in the sense that sets of λ-measure zero must also have ν_n^a-measure zero. This property is known as *absolute continuity* of ν_n^a with respect to λ. Additionally, mutations must increasingly focus within the partitions $\{E_n^a\}_{a \in A_n}$.

Definition 11.3.5. *The mutation distributions* ν_n^a *are increasingly focused within a sequence of partitions* $\{E_n^a\}_{a \in A_n}$ *if* $\nu_n^a(E_n^a) \to 1$ *as* $n \to \infty$.

This requirement of increasing focus is most easily satisfied by construction. That is, the convergence $\nu_n^a(E_n^a) \to 1$ can be built into the definition of ν_n^a by tying the variance of the mutation distribution to the size of E_n^a, as will be done in Section 8.1 below. For instance, the choice of $\nu_n^a(B) = \lambda(B \cup E_n^a)/\lambda(E_n^a)$ is increasingly focused.

In addition, each partition must be such that the partition point is contained in the interior of the partition. Partitions that satisfy this requirement will be described as *padded* because there is at least one open set within the partition containing the partition point.

Definition 11.3.6. *A partition* $\{E^a\}_{a \in A}$ *with* $a \in E^a$ *for all* $a \in A$ *is padded if for each* $a \in A$ *there is an open set* O^a *with* $a \in O^a$ *and* $O^a \subseteq E^a$.

An evolutionary annealing algorithm produces *padded partitions* if the partitions generated by its separation routine are padded for all n. A sequence of padded partitions yields sequences of sets that shrink nicely around each mixing point.

Finally, the cooling schedule and mutation variance must decay slowly enough to guarantee full exploration of the search space, or else the global optimum might be missed. Note that this requirement pertains to the shape of the mutation distributions and not just the variance. Specifically, let $E \subseteq X$ be an open region of the search space with positive λ-measure, and let E^c be its complement in X. Recall that A_n represents the set of all previously observed individuals at time n. Ultimately, to fully explore the space, there must be an n such that $A_n \cap E \neq \emptyset$ for each open set E. As a technical detail, X must be separable in order for this to be possible, in which case X has a countable dense subset. If the space is fully explored, then $A_n \uparrow A$ where A is a countable dense subset of X. An evolutionary annealing algorithm that satisfies this criterion will be termed *exhaustive*.

Intuitively, in order to be exhaustive, an evolutionary annealing algorithm must not sharpen too quickly; that is, both the variance and the temperature must decay in such a way that every open region is traversed at some generation with probability 1. Let $\mathbb{G}_n(E) = \sum_{a \in A_n} p_n(a) \nu_n^a(E)$ be the probability that a single sample from evolutionary annealing lies inside an open region $E \subseteq X$ at time n. Define $\gamma_{n,K}$ to be the probability of sampling a point in E from $\mathbb{G}_n$ on at least one of the K samples in the population at time n, noting that $\gamma_{n,K} > \mathbb{G}_n(E)$. Let $\alpha_1 = \gamma_{1,K}$ be the probability of having encountered E in the first generation and recursively define $\alpha_n = \gamma_{n,K}[1 - \alpha_{n-1}]$ so that α_n gives the probability of having encountered E by the n^{th} generation. Then the algorithm is exhaustive whenever $\sum_{n=1}^{\infty} \alpha_n = 1$ for each open set E. Unfortunately, it is not currently known what properties make an evolutionary annealing algorithm exhaustive. However, larger sample sizes, slower cooling, and slower mutation decay should contribute towards making an algorithm exhaustive for many fitness functions. The study of such properties is left for future work.

Notice that the requirement that the mutation distributions be increasingly focused and that the algorithm be exhaustive are in competition with each other. Increasing focus requires the mutation variance to shrink quickly enough, and exhaustiveness requires it not to shrink to quickly.

The convergence theorem can now be stated. For convergence, evolutionary annealing is assumed to use the partitioning method to set the region weights $c_n(a)$. The proof makes liberal use of the partitions $\{E_n^a\}$.

Theorem 11.3.1. *An exhaustive evolutionary annealing algorithm with annealed proportional selection converges in probability to the minimal points of any suitable fitness function provided that it produces padded partitions and its mutation distributions are increasingly focused within the partitions.*

Proof. Fix $\epsilon, \delta > 0$ with $\epsilon < \gamma$. Without loss of generality, assume $\lambda(X) = 1$; if not, $\tilde{\lambda} \equiv \lambda / \lambda(X)$ will satisfy this equality. Let $\mathbb{G}_n(B) = \sum_{a \in A_n} p_n(a) \nu_n^a(B)$ be the distribution generating evolutionary annealing at time n. The desired result will follow if there exists an N such that for $n \geq N$, $\mathbb{G}_n(X_\epsilon) > 1 - \delta$. Because $\mathcal{A}(X_\epsilon) = 1$ for all ϵ, it is sufficient to prove that $|\mathbb{G}_n(X_\epsilon) - \mathcal{A}(X_\epsilon)| < \delta$ for large n .

The conclusions below will require the sequence $F_n^a = X_\epsilon \cap E_n^a$ to shrink nicely around a. The interior of X_ϵ has positive λ-measure since f is suitable, and so it suffices for E_n^a to shrink nicely. Because the algorithm is exhaustive, any open set containing a must eventually contain E_n^a for $n > N$ since $A_n \uparrow A$ with A dense in X. Because the partition E_n^a is padded, a is contained in the interior of E_n^a, which is an open set with positive measure. Thus F_n^a shrinks nicely to a.

For convenience, let $\lambda_n^a(X_\epsilon) \equiv \lambda(X_\epsilon \cap E_n^a) / \lambda(E_n^a)$. Define $\tilde{\mathbb{G}}_n$ so that $\tilde{\mathbb{G}}_n(X_\epsilon) = \sum_{a \in A_n} p_n(a) \lambda_n^a(X_\epsilon)$. Since ν_n^a is increasingly focused within E_n^a, for n sufficiently large, $\nu_n^a(X_\epsilon \setminus E_n^a) < \delta/4$.

Also, because the algorithm is exhaustive, there exists n large enough so that either $\mathring{X}_\epsilon \cap E_n^a = E_n^a$ or $\mathring{X}_\epsilon \cap E_n^a = \emptyset$ where as usual $\mathring{X}_\epsilon \equiv X_\epsilon \setminus \partial X_\epsilon$. Since $\epsilon < \gamma$, the measure of the boundary of X_ϵ can be ignored, and either $\lambda_n^a(X_\epsilon) = 0$ or $\lambda_n^a(X_\epsilon) = 1$. Similarly, $\nu_n^a(X_\epsilon \cap E_n^a)$ can be chosen to be within $\delta/4$ of either 0 or 1, since ν_n^a is increasingly centered on a and either $\nu_n^a(X_\epsilon \cap E_n^a) = \nu_n^a(E_n^a)$ or $\nu_n^a(X_\epsilon \cap E_n^a) = 0$, depending on whether $a \in \mathring{X}_\epsilon$. Therefore,

$$\begin{aligned} \left| \mathbb{G}_n(X_\epsilon) - \tilde{\mathbb{G}}_n(X_\epsilon) \right| &\leq \sum_{a \in A_n} p_n(a) \left| \nu_n^a(X_\epsilon) - \lambda_n^a(X_\epsilon) \right| \\ &\leq \sum_{a \in A_n} p_n(a) \nu_n^a(X_\epsilon \setminus E_n^a) \\ &\quad + \sum_{a \in A_n} p_n(a) \left| \nu_n^a(X_\epsilon \cap E_n^a) - \lambda_n^a(X_\epsilon) \right| \\ &< \frac{\delta}{4} + \frac{\delta}{4} = \frac{\delta}{2}. \end{aligned} \tag{11.16}$$

Thus $|\mathbb{G}_n(X_\epsilon) - \tilde{\mathbb{G}}_n(X_\epsilon)| < \frac{\delta}{2}$.

Next it will be shown that $|\tilde{\mathbb{G}}_n(X_\epsilon) - \mathcal{A}(X_\epsilon)| \to 0$. The argument is based on the fact that $\tilde{\mathbb{G}}_n$ is an approximate martingale and uses a series of conditional expectations. Here and below, the notation $1_B = 1_B(x)$ is defined as

$$1_B(x) = \begin{cases} 1 & \text{if } x \in B \\ 0 & \text{otherwise} \end{cases} . \tag{11.17}$$

The fact that $\int_X 1_B d\lambda = \lambda(B)$ is used frequently below. The expression $Y1_B$ is shorthand for $Y(x)1_B(x)$, as is common in the study of stochastic processes.

Let $Y_n(x) = \sum_{a\in A_n} 1_{E_n^a}(x) g_n(a)$ be a random process on the same space. Next, it will be shown that $\left|\mathbb{E}(Y_n 1_{X_\epsilon}) - \tilde{\mathbb{G}}_n(X_\epsilon)\right| < \frac{\delta}{4}$. Let $h_n(x) = \exp(-f(x)/T_n)$, so that $\xi_n = \sum_{a\in A_n} h_n(a)$ from Equation 11.4. Define $\eta_n = \int_X h_n(z)\lambda(dz)$. Also, h_n is approximated by its neighborhood average since $h_n(x) = \eta_n g_n(x)$ and g_n is approximated by its neighborhood average. Then

$$|p_n(a) - g_n(a)\lambda(E_n^a)| \;=\; h_n(a)\,\lambda(E_n^a)\,\frac{|\xi_n - \eta_n|}{\xi_n \eta_n}. \tag{11.18}$$

For any $\beta > 0$ and n large,

$$\begin{aligned} |\xi_n - \eta_n| &= \left|\sum_{x\in A_n} h_n(x)\lambda(E_n^x) - \int_X h_n(z)\lambda(dz)\right| \\ &\le \sum_{x\in A_n} \left|h_n(x)\lambda(E_n^x) - \int_{E_n^x} h_n(z)\lambda(dz)\right| \\ &= \sum_{x\in A_n} \left|h_n(x) - h_n^\lambda(x)\right| \lambda(E_n^a) < \beta. \end{aligned} \tag{11.19}$$

Therefore $|p_n(a) - g_n(a)\lambda(E_n^a)| \to 0$, and so

$$\begin{aligned} \left|\tilde{\mathbb{G}}(X_\epsilon) - \mathbb{E}(Y_n 1_{X_\epsilon})\right| &= \left|\sum_{a\in A_n}\left[p_n(a)\lambda_n^a(X_\epsilon) - \int_X 1_{X_\epsilon\cap E_n^a}(z) g_n(a)\lambda(dz)\right]\right| \\ &\le \sum_{a\in A_n} |p_n(a) - g_n(a)\lambda(E_n^a)|\;\lambda_n^a(X_\epsilon) < \frac{\delta}{4}. \end{aligned} \tag{11.20}$$

Let $\{\mathcal{E}_n^{A_n}\}$ be the filtration generated by the sequence of partitions $\{E_n^a\}$. Now consider the process generated by conditioning $Y_n 1_{X_\epsilon}$ on $\{\mathcal{E}_n^{A_n}\}$:

$$\tilde{Y}_n^\epsilon(x) \;= \mathbb{E}\left(Y_n(x) 1_{X_\epsilon}(x) \mid \mathcal{E}_n^{A_n}\right) \;= \sum_{a\in A_n} 1_{E_n^a}(x) g_n(a)\,\lambda(X_\epsilon \cap E_n^a).$$

Note that $\mathbb{E}(\tilde{Y}_n^\epsilon) = \mathbb{E}\left(Y_n 1_{X_\epsilon}\right)$ by the properties of conditional expectations. It is also the case that $\tilde{Y}_n^\epsilon$ converges to $\mathcal{A}\left(X_\epsilon\right)$ because for $\eta > 0$,

$$\begin{aligned}
\left|\mathcal{A}\left(X_\epsilon\right) - \tilde{Y}_n^\epsilon\right| &\leq \sum_{a \in A_n} 1_{E_n^a} \left|\mathcal{A}(X_\epsilon \cap E_n^a) - g_n\left(a\right) \lambda\left(X_\epsilon \cap E_n^a\right)\right| \\
&\leq \frac{\eta}{3} + \sum_{a \in A_n} 1_{E_n^a} \left|\mathcal{A}_n(X_\epsilon \cap E_n^a) - g_n\left(a\right) \lambda\left(X_\epsilon \cap E_n^a\right)\right| \\
&= \frac{\eta}{3} + \sum_{a \in A_n} 1_{E_n^a} \left|\int_{X_\epsilon \cap E_n^a} g_n(x)\, \lambda(dx) - g_n\left(a\right) \lambda\left(X_\epsilon \cap E_n^a\right)\right| \\
&= \frac{\eta}{3} + \sum_{a \in A_n} 1_{E_n^a} \int_{X_\epsilon \cap E_n^a} \left|g_n\left(x\right) - g_n\left(a\right)\right| \lambda\left(dx\right). \qquad (11.21)
\end{aligned}$$

In the previous equations, $\mathcal{A}$ was approximated $\mathcal{A}_n$ and definitions were applied. If g_n were continuous, then the equations above would complete the proof. Since g_n may be discontinuous, the neighborhood average g_n^λ can be inserted into the difference $|g_n(x) - g_n(a)|$ to obtain

$$\begin{aligned}
\left|\mathcal{A}\left(X_\epsilon\right) - \tilde{Y}_n^\epsilon\right| &\leq \frac{\eta}{3} + \sum_{a \in A_n} 1_{E_n^a} \int_{X_\epsilon \cap E_n^a} \left|g_n^\lambda\left(x\right) - g_n\left(a\right)\right| \lambda\left(dx\right) \\
&\quad + \sum_{a \in A_n} 1_{E_n^a} \int_{X_\epsilon \cap E_n^a} \left|g_n\left(x\right) - g_n^\lambda\left(x\right)\right| \lambda\left(dx\right) \\
&\leq \frac{\eta}{3} + \sum_{a \in A_n} 1_{E_n^a} \left[\sup_{x \in E_n^a} \left|g_n^\lambda\left(x\right) - g_n\left(a\right)\right|\right] \lambda\left(E_n^a\right) + \frac{\eta}{3} \\
&< \frac{\eta}{3} + \frac{\eta}{3} + \frac{\eta}{3} = \eta. \qquad (11.22)
\end{aligned}$$

The inequalities hold because g_n is approximated by its neighborhood average at a, and $X_\epsilon \cap E_n^a$ shrinks nicely. It follows that $\mathbb{E}\left(Y_n 1_{X_\epsilon}\right) = \mathbb{E}\tilde{Y}_n^\epsilon \to \mathcal{A}\left(X_\epsilon\right)$. That is,

$$\begin{aligned}
\left|\tilde{\mathbb{G}}_n\left(X_\epsilon\right) - \mathcal{A}\left(X_\epsilon\right)\right| &\leq \left|\tilde{\mathbb{G}}_n\left(X_\epsilon\right) - \mathbb{E}\left(Y_n 1_{X_\epsilon}\right)\right| + \left|\mathbb{E}\left(Y_n 1_{X_\epsilon}\right) - \mathcal{A}\left(X_\epsilon\right)\right| \\
&< \frac{\delta}{4} + \frac{\delta}{4} = \frac{\delta}{2}. \qquad (11.23)
\end{aligned}$$

Putting it together, for n sufficiently large,

$$\begin{aligned} \left|\mathbb{G}_n\left(X_\epsilon\right)-\mathcal{A}\left(X_\epsilon\right)\right| &\leq \left|\mathbb{G}_n\left(X_\epsilon\right)-\tilde{\mathbb{G}}\left(X_\epsilon\right)\right|+\left|\tilde{\mathbb{G}}_n\left(X_\epsilon\right)-\mathcal{A}\left(X_\epsilon\right)\right| \\ &< \frac{\delta}{2}+\frac{\delta}{2}=\delta, \end{aligned} \tag{11.24}$$

completing the proof. □

As a corollary, annealed tournament selection also converges.

Corollary 11.3.2. *An exhaustive evolutionary annealing algorithm with annealed tournament selection converges in probability to the global minima of any suitable fitness function provided that the mutation distributions are increasingly focused within the partition.*

Proof. The proof will follow by recasting the annealed tournament selection probability from Equation 11.9 as a problem for annealed proportional selection with an altered fitness function and cooling schedule. Let f be a suitable fitness function. Let $A = \lim_n A_n$ be a countable, dense subset of X. Define $r(x)$ to enumerate the members of A according to their rank on f starting at 0 for the minimal point, with ties broken according to any deterministic scheme. Define $r_n(x)$ to provide a similar enumeration of $A_n$with the same tie-breaking procedure, and extend $r_n(x)$ to all of A by setting $r_n(x) = |A_n|$ on $x \in A \setminus A_n$. Then $r(x) = \lim_n r_n(x)$.

Given selection pressure q, define $h\left(x\right) = \left(1-q\right)^{r(x)}$ on the set A, and let $h_n\left(x\right) = \left(1-q\right)^{r_n(x)}$. The function h will serve as the basis for a convergence problem with annealed proportional selection after converting the cooling schedule to pull the temperature exponent to the outside. Define $\tilde{T}_n \equiv \frac{\log 1-q}{\log 1-q^{1/T_n}}$ so that $\left(1-q^{1/T_n}\right) = \left(1-q\right)^{1/\tilde{T}_n}$. It is possible to do so because q is fixed. Then

$$h\left(x\right)^{1/\tilde{T}_n} = \left(1-q\right)^{r(x)/\tilde{T}_n} = \left(1-q^{1/T_n}\right)^{r(x)}. \tag{11.25}$$

As a final step, let $u(x) = -\log h(x)$, and then

$$\exp\left(-\frac{u(x)}{\tilde{T}_n}\right)\lambda\left(E_n^x\right) \;=\; h(x)^{1/\tilde{T}_n}\lambda\left(E_n^x\right) \;\propto\; q^{1/T_n}\left(1-q^{1/T_n}\right)^{r(x)}\lambda\left(E_n^x\right). \tag{11.26}$$

That is, if the ranking function is fixed at r, then there is a function u such that proportional selection with u and the cooling schedule $\tilde{T}_n$ is equivalent to tournament selection with the ranking function r and cooling schedule T_n. If the convergence result of Theorem 11.3.1 holds for u, then it holds for h as well.

In order to apply Theorem 11.3.1, it is necessary to show that the conditions are met. First, u must be suitable whenever f is. Now since $f^* > -\infty$, the enumeration r is well-founded on A; say $r(z) = 0$ for some $z \in A$ with $r(y) > r(z)$ for all $y \neq x$. Next, u must be extended to all of X. To do this, choose any continuous extension of r to all of X. The resulting extension of u automatically satisfies the neighborhood approximation requirement. Also, $0 \leq h(x) \leq 1$ because $q \in [0, 1]$. The measure λ was chosen to be finite, so $\int_X h(x)^{1/\tilde{T}_n}\, d\lambda \leq \lambda(X)$, i.e., u possesses the necessary integrability properties. Since the σ-algebra $\mathcal{F}$ was presumed to support the partitions, the sets X_ϵ^u for u are $\mathcal{F}$-measurable, and due to the continuity of u and the fact that f is suitable, $\lambda\left(\partial X_\epsilon^u\right) = 0$ and $\lambda\left(X_\epsilon^u\right) > 0$. Thus u is a suitable fitness function, and the convergence theorem holds.

The final issue to complete the proof is to show that iteratively ranking the population with r_n does not undo the convergence result. To see this fact, set $p_n(x) = \xi_n h(x)^{1/\tilde{T}_n} \lambda\left(E_n^x\right)$ and set $p_n^m(x) = \xi_n^m h_n(x)^{1/\tilde{T}_n} \lambda\left(E_n^x\right)$ with ξ_n and ξ_n^m as normalizing factors. Observe that $\lim_{m\to\infty} p_n^m(x) = p_n(x)$. Now define X_ϵ and $\mathbb{G}_n(B)$ as in Theorem 11.3.1, and define $\mathbb{G}_n^m(B) = \sum_{a\in A_n} p_n^m(a) \nu_n^a(B)$. Fix $\delta > 0$. Then for m sufficiently large,

$$\begin{aligned} \left|\mathbb{G}_n^m(X_\epsilon) - \mathbb{G}_n(X_\epsilon)\right| &\leq \sum_{a\in A_n} \left|p_n^m(a) - p_n(a)\right| \nu_n^a(X_\epsilon) \\ &< \sum_{a\in A_n} \delta 2^{-n} \nu_n^a(X) < \delta. \end{aligned} \tag{11.27}$$

As a result, for all n greater than the requisite m, $\left|\mathbb{G}_n^n(X_\epsilon) - \mathbb{G}_n(X_\epsilon)\right| < \delta$. Notice that $\mathbb{G}_n^n$ is exactly evolutionary annealing with annealed tournament selection on f with cooling schedule T_n, and that $\mathbb{G}_n$ is evolutionary annealing with annealed proportional selection on u with cooling schedule $\tilde{T}_n$. Therefore evolutionary annealing with annealed tournament selection converges in probability given the assumptions. □

In sum, these theorems show that evolutionary annealing is guaranteed to converge asymptotically arbitrarily close to the global minima of the fitness function provided that the cooling schedule and variance decay are not too aggressive. The theorems do not say much about the rate of convergence, but certain heuristic principles can be identified, as will be discussed next.

11.3.2 Convergence Rates

An examination of the proof of Theorem 11.3.1 shows that there are three basic sources of approximation error: (1) the variance of the mutation distribution, (2) the accuracy of the neighborhood average, and (3) the speed of convergence for the annealing distributions, due to the cooling schedule. Of these, the variance and the cooling schedule are under the direct control of the practitioner. Implicitly, these two factors also control the accuracy of the neighborhood average. In order to set the cooling schedule and variance decay to maximize the rate of convergence, the effects of these three error sources must be carefully considered.

The first source of error is due to the difference $|\nu_n^a(X_\epsilon) - \lambda_n^a(X_\epsilon)|$. Convergence occurs because both of these measures ν_n^a and λ_n^a asymptotically become point masses, the former because variance decays and the latter because the mixing points eventually fill the search space. To minimize error, these two measures should be kept as close as possible for mixing points in the vicinity of the optima. As the algorithm begins to focus on a small group of optima, the partitions in that region will become smaller, and the variance of the mutation distribution should decrease at a similar rate. Notably, however, decreasing the variance also reduces the probability that the global optimum will be discovered if it has not already and if it is located sufficiently far from the current regions of focus. Also, when a new and better local optimum is discovered after the search has already focused on other local optima, the exploration of the new optimum will proceed slowly if the variance has already decayed substantially. Therefore it may make sense to scale the variance to reflect the size of the partition region for the mixing point being mutated. In this way, larger variances will typically be employed in unexplored regions, whereas a narrower variance will be used in well explored regions, so that ν_n^a

and λ_n^a are well-matched in general. The fact that the mixing points eventually fill the space guarantees that a dynamic and locally scaled variance decay schedule of this type will eventually become increasingly centered as required.

The second source of error pertains to the accuracy of approximating the average value of the annealing distribution by its value at the mixing points, due to the term $\left|g_n^\lambda(x) - g_n(a)\right|$. This error depends strongly on the fitness function. If the fitness function is relatively homogeneous and does not fluctuate at different rates in different regions of the search space, then this source of error strongly reflects the mass of the partition region, $\lambda\left(E_n^a\right)$. In a large region, the approximation $g_n(a)\lambda\left(E_n^a\right)$ is likely to differ more substantially from $\int_{E_n^a} g_n \, d\lambda$ than it would in a smaller region. Thus this source of error can perhaps be reduced by spreading the mixing points more evenly through the space in order to keep the partition regions uniformly small. This goal can be accomplished by using a high variance at the outset to guarantee full exploration of the space. At later stages, the use of a high variance is at odds with the need to match the mutation distribution ν_n^a with the measure λ_n^a, and thus the first two sources of error must be balanced and cannot be mutually eliminated. The accuracy of the average approximation is also affected by the cooling schedule. At high temperature, the approximation must be more accurate because g_n will vary less over the region E_n^a if T_n is large. Lowering the temperature increases this source of error by causing the function g_n to fluctuate more.

The third source of error concerns the speed of convergence of the annealing distributions due to the difference $|g - g_n|$. The faster the cooling schedule takes the temperature to zero, the faster this error will be minimized. But a fast cooling schedule will increase the error due to the neighborhood approximation. The speed of the cooling schedule must balance the need to minimize both the neighborhood approximation error and the annealing convergence error.

Overall, preliminary experiments showed that a logarithmic cooling schedule, e.g. $T_n^{-1} = \eta \log n$, works well in practice. Early results also suggest that the variance should start off quite large and decay exponentially fast. Also, scaling the variance locally based on the size of the partition region for the mixing point being mutated should be effective. The next chapter discusses experimental results along these lines.

11.4 Conclusion

Evolutionary annealing was introduced as an optimization strategy that seeks the global optimum by building a probabilistic model based on the objective evaluations of all points produced by the optimization process. Under certain conditions, this model converges to an accurate representation of the objective function near the global optimum, as seen in Theorem 11.3.1. The proof relies on the fact that the model underlying evolutionary annealing is an approximate Levy martingale. Such a martingale may be thought of as an estimate of some quantity that improves as more information becomes available. Thus evolutionary annealing is a *martingale method*, a new class of optimization method based on using increasing information to improve optimization. In the case of evolutionary annealing, objective evaluations provide the source of information, and evolutionary annealing leverages this information in order to improve its optimization.

Evolutionary annealing converges in theory, but its experimental performance also needs to be analyzed. To do so, evolutionary annealing must be instantiated within a particular search domain. Chapter 12 presents experiments with an instantiation in finite-dimensional Euclidean space, $\mathbb{R}^d$, along with specific implementation details that make it possible to sample evolutionary annealing in logarithmic time with respect to the number of evaluations. In the course of these experiments, it is seen that evolutionary annealing generally performs well. Chapter 13 applies evolutionary annealing to the problem of training neural networks, showing that the basic optimization concept can be expanded to search effectively in complex spaces. These neural networks outperform networks trained by other methods on tasks that require a complex network topology. Successes in both real vectors and neural networks will establish evolutionary annealing as an effective optimization method for practical tasks.

Chapter 12

Evolutionary Annealing In Euclidean Space

Evolutionary annealing was developed in the last chapter as a general-purpose optimization technique. This chapter presents an application of evolutionary annealing to the space of finite real vectors. Experiments are performed to compare real-space evolutionary annealing (REA) on the set of benchmarks and algorithms from Chapter 8. REA performs well in general and is complementary to the earlier optimizers tested, outperforming other methods on multimodal objectives with irregular structure. This feature suggests that REA is well-aligned with a mixture sieve prior.

12.1 Evolutionary Annealing in Euclidean Space

Evolutionary annealing can be used to search for bit strings, real vectors, neural networks, Bayesian network structures, game strategies, programs, state machines, and any other structure that can be embedded within a suitable measure space. As a baseline evaluation, experiments were performed in finite-dimensional Euclidean space on a set of twelve standard benchmarks from Chapter 8. As before, the domain was a problem-specific hypercube $Q \subseteq \mathbb{R}^d$, with a normalized Lebesgue measure on the Borel σ-algebra restricted to Q, i.e. $\lambda(B) = \int_B dx / \int_Q dx$. The instantiation of evolutionary annealing in Euclidean space is termed Real-Space Evolutionary Annealing (REA), and it was tested with both annealed proportional selection (REA-P) and annealed tournament selection (REA-T).

12.1.1 Instantiation Details

The version of REA implemented for this article uses Gaussian mutation distributions with $\nu_n^a = \mathcal{N}\left(a, \sigma_n\left(a\right)^2\right)$. The standard deviation $\sigma_n(a)$ is scaled to the area of the partition region with $\sigma_n(a) = \frac{1}{2} w \lambda \left(E_n^a\right)^{1/d}$, where d is the dimension of the problem and w is the width of the space (i.e. $\frac{1}{2}w$ is the side length of Q). This choice of variance seeks to align the shape of ν_n^a and $\lambda\left(E_n^a\right)$ as discussed in Section 11.3.2. Specifically, if E_n^a were a hypercube, then the first standard deviation of ν_n^a would be contained within E_n^a.

This implementation of REA does not meet the requirements of Theorem 11.3.1, but it is on the cusp of doing so. Most importantly, there is no way to determine at this time whether REA is exhaustive, though it may be if a small enough learning rate is used. The vector-separating algorithm from Section 11.2.3 was used, and it does produce padded partitions in $\mathbb{R}^d$. The mutation distributions are absolutely continuous with respect to the Lebesgue measure. On average, $\nu(E_n^a)$ is a nonzero constant less than 1. If the partitions were to reduce in size regularly, and if $\sigma_n(a)$ were multiplied by a decaying factor, say, $e^{-\beta n}$ for β close to zero, then ν_n^a would be increasingly focused ($\nu_n^a(E_n^a) \to 1$). In the experiments that follow, no decay factor was applied for $d = 5$ and $d = 10$. It was not necessary to do so, since the samples in the experiments converged towards a fixed distribution without a decay factor. In 25 dimensions, however, a decay factor of $n^{-\frac{1}{2}}$ (i.e. $\sigma_n(a) = \frac{1}{2} w n^{-\frac{1}{2}} \lambda\left(E_n^a\right)^{1/d}$) was applied in order to achieve faster convergence.

REA can be computationally expensive because of the overhead involved in handling an expanding set of evaluations points. An efficient implementation can be obtained by implementing the sampling routines in a way that requires computation that is only logarithmic in the number of evaluation point. The details are discussed next.

12.1.2 Sampling Algorithms for Annealed Selection

The computational efficiency of evolutionary annealing is primarily determined by the cost of preparing and sampling annealed proportional selection. A naïve approach to computing Equation 11.4 would make the cost of preparing and sampling p_n be linear, since the normalizing factor ξ_n must be computed one element at a time and because sampling from a probability vector typically requires iterating through the vector. In fact, annealed proportional selection can be approximately computed in logarithmic time in the average case by leveraging the partition tree, with most operations occurring in subroutines that guarantee worst-case logarithmic time, to be described in Section 11.2.4. The approximation can be made accurate at close to machine-level precision, so that it is sufficiently precise for all practical purposes.

In order to reduce the sampling complexity for evolutionary annealing from linear to logarithmic time, a tree-sampling method is needed for sampling p_n. The partition tree provides a tree such that the leaves are associated exactly with the components of p_n. The goal, then, is to create a sequence of decisions made along a path through the partition tree such that the decision process assigns probability mass to each complete path in equality with the probability of the leaf at the end of the path under p_n.

Let ν be an internal node of the partition tree. Let $N \subseteq A_n$ be the set of previously observed individuals residing within leaves of the partition tree that are descended from ν. Let μ be one of the two child nodes of ν, and let $M \subseteq N$ contain the leaf descendants of μ. To extend a probabilistic path that has reached ν, a choice must be made at node ν whether to add node μ or its sibling to the path. Suppose the choice is made according to

$$\mathbb{P}\left(\mu \mid \nu\right) = \frac{\sum_{x\in N} \alpha(x)^{\log n} \lambda\left(E_n^x\right)}{\sum_{y \in M} \alpha(y)^{\log n} \lambda\left(E_n^y\right)}, \tag{12.1}$$

where $\alpha(x) \equiv \exp\left(-\eta f(x)\right)$, mirroring Equation 11.4 with cooling schedule $T_n^{-1} = \eta \log n$. Now let π_x be a path from the root to the leaf containing the point x, and observe that a sequence of decisions made according to Equation 12.1 yields

$$\mathbb{P}\left(\pi_x\right) = \prod_{\nu} \mathbb{P}\left(\text{child}\left(\nu, \pi_x\right) \mid \nu\right) = \xi_n \alpha(x)^{\log n} \lambda\left(E_n^x\right) = p_n\left(x\right), \tag{12.2}$$

with $\text{child}\left(\nu, \pi_x\right)$ being the child node of ν on the path π_x. The next to last equality in Equation 12.2 holds because each successive denominator cancels the numerator of the previous one, leaving only the denominator from the root node, which is equal to ξ_n^{-1}, and the numerator from the leaf node, which is $\alpha(x)^{\log n}\lambda\left(E_n^x\right) = \exp\left(-f(x)/T_n\right)\lambda\left(E_n^x\right)$. Therefore, sampling a path through the tree starting from the root samples from p_n provided that the decision at each node is made according to Equation 12.1.

The difficulty of this method is that the sum in the numerator of Equation 12.1 must be computed for each node. If the temperature were fixed, then the value of the sum could be stored on each node. The sum only changes when new leaves are inserted, and then only the nodes that are direct ancestors of the inserted node need to adjust their sums, resulting in logarithmic updates to the tree. As long as the temperature does not change, then, the tree-sampling method is logarithmic both to prepare the data structures and to sample them.

It remains to account for changes in temperature without recomputing the numerator of Equation 12.1 at each time step. Introducing $h(T) = \sum_{x\in N} \alpha(x)^T \lambda\left(E_n^x\right)$ to capture the fact that the sum varies with the generation, the problem is that the exponent cannot be pulled out of the sum, meaning that the sum must be recomputed with every change in temperature. However, $h(T)$ is infinitely differentiable in T, with m^{th} derivative

$$h^{(m)}(T) = \sum_{x\in N} \alpha(x)^T \left(\log\alpha(x)\right)^m \lambda\left(E_n^x\right). \tag{12.3}$$

Thus a Taylor approximation is possible, since

$$h(T) = \sum_{m=1}^{\infty} \left(\sum_{x\in N} \frac{\left(\log\alpha(x)\right)^m}{m!} \alpha(x)^{T_0} \lambda\left(E_n^x\right) \right) \left(T - T_0\right)^m. \tag{12.4}$$

The Taylor approximation can be computed by storing a vector of coefficients $t = (t_1 \ldots t_m)$ with $t_j \equiv \sum_{x\in N} \left(\log\alpha(x)\right)^j \alpha(x)^{T_0} \lambda\left(E_n^x\right)$ for all $j \in 1 \ldots m$, with a fixed value T_0. These vector sums can then be propagated up the tree in logarithmic time, and the sampling method can approximate $h\left(\log n\right)$ as needed at each node.

To complete the description of the sampling method, T_0 and m must be specified. As a general feature of $h(T)$, the approximation is substantially correct for $T > T_0$ over a larger interval than for $T < T_0$. With $m = 10$, the approximation is highly accurate for $T \in [T_0, T_0 + 1/2]$ but degrades outside that interval. Thus the Taylor coefficients must be recomputed for the entire tree on every interval of T of size 1/2. For practical purposes, the value of T_0 is set to 1 for the first few generations, and then is reset when

$$T = \log n = 3/2, 2, 5/2, \ldots$$

This resetting feature is actually not as burdensome as it may sound, and it only needs to be performed logarithmically often, so that the entire procedure of maintaining and sampling the tree still has logarithmic complexity overall. Some example statistics for computation time are shown in Table 12.1. The next section discusses a similar method for sampling annealed tournament selection, and introduces data structures that make it possible to sample annealed selection in average case logarithmic time.

12.1.3 Sampling Annealed Tournament Selection

As with annealed proportional selection, it is not computationally efficient to sample Equation 11.9 directly. In addition, annealed tournament selection introduces the need to sort all previously proposed solutions by fitness. In order to accommodate these issues, a balanced binary tree can be used, called the *score tree*. Like the partition tree, the score tree contains one leaf node per proposed solution; the internal nodes represent the set of nodes in their span. The score tree reorganizes the partition tree so that points with higher fitness are always to the left and points with lower fitness are always to the right. Using standard tree algorithms, the score tree can be balanced in logarithmic time after each insertion.

Annealed ranking selection can be sampled by walking the score tree, making a decision at each node whether to follow the lower- or the higher-ranked branch. The probability at each node will depend on the area represented by the node and the height of the subtree underneath the node. The area of a leaf node can be copied from the partition tree. Both the area and the height can then be propagated up the score tree in logarithmic time after each insertion. In this way, the score tree is also a partition tree. However, the internal nodes of the score tree correspond approximately to the level sets of the fitness function, and thus the regions that they represent can be arbitrarily complex to describe. Therefore, although the score tree defines a partition over the search space, the score tree cannot replace the partition tree, because there is no efficient way to determine whether a point resides in the region represented by an internal node of the score tree. However, the score tree is kept balanced, providing worst-case logarithmic performance.

When sampling annealed tournament selection using the score tree, the decision must be made at each internal node ν whether to follow the higher- or lower-ranked branch. Let $h+1$ be the height of the subtree under node ν, and assume the tree is perfectly balanced. Then ν has 2^{h+1} leaf nodes in its span. Let μ be the higher-ranked child node of ν. Suppose further that the nodes spanned by ν range in rank from R to $R+2^{h+1}-1$, so that the nodes spanned by μ range in rank from R to $R+2^h-1$. Ignoring the region weight temporarily, a direct application of standard tournament selection yields

$$\mathbb{Q}_T\left(\mu \mid \nu\right) = \frac{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+m}}{\sum_{j=0}^{2^{h+1}-1} q^{1/T}\left(1-q^{1/T}\right)^{R+j}}. \tag{12.5}$$

Let κ be the lower ranked sibling of μ, spanning ranks $R+2^h$ to $R+2^{h+1}-1$. Then the ratio for selecting μ over κ is given by

$$\frac{\mathbb{Q}_T\left(\mu \mid \nu\right)}{\mathbb{Q}_T\left(\kappa \mid \nu\right)} = \frac{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+m}}{\sum_{m=0}^{2^h-1} q^{1/T}\left(1-q^{1/T}\right)^{R+2^h+m}} = \frac{1}{\left(1-q^{1/T}\right)^{2^h}} \equiv \tilde{q}\left(h,T\right). \tag{12.6}$$

The function $\tilde{q}(h,T)$ gives the selection preference of the higher branch over the lower branch. Finally, incorporating the region weights, let

$$\mathbb{P}_T\left(\mu \mid \nu\right) = \frac{\tilde{q}\left(h,T\right)\lambda\left(\mu\right)}{\tilde{q}\left(h,T\right)\lambda\left(\mu\right) + \left(1-\tilde{q}\left(h,T\right)\right)\lambda\left(\kappa\right)}, \tag{12.7}$$

where $\lambda\left(\mu\right)$ and $\lambda\left(\kappa\right)$ are the cumulative weights of the partition regions of the points in the span of μ and κ, respectively. Equation 12.7 is normalized and implies $\mathbb{P}_T\left(\kappa \mid \nu\right) = 1 - \mathbb{P}_T\left(\mu \mid \nu\right)$.

To show that this process does in fact implement annealed tournament selection, notice that

$$\mathbb{P}_T\left(\mu \mid \nu\right) \propto \tilde{q}\left(h,T\right)\frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)}\ ,\ \ \mathbb{P}_T\left(\kappa \mid \nu\right) \propto \frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)}, \tag{12.8}$$

introducing the $\lambda(\nu)$ factor as a proportional constant. Thus for a general path π_x, recalling that $\tilde{q}(h,T) \propto \mathbb{Q}_T\left(\mu \mid \nu\right)$ by definition,

$$\mathbb{P}_T\left(\text{child}\left(\nu,\pi_x\right) \mid \nu\right) \propto \mathbb{Q}_T\left(\text{child}\left(\nu,\pi_x\right) \mid \nu\right)\frac{\lambda\left(\mu\right)}{\lambda\left(\nu\right)}, \tag{12.9}$$

and therefore

$$\begin{aligned}
\mathbb{P}_{T_n}\left(\pi_x\right) &= \prod_{\nu \in \pi_x} \mathbb{P}_{T_n}\left(\operatorname{child}\left(\nu, \pi_x\right) \mid \nu\right) \\
&\propto \prod_{\nu \in \pi_x} \mathbb{Q}_{T_n}\left(\operatorname{child}\left(\nu, \pi_x\right) \mid \nu\right) \frac{\lambda\left(\operatorname{child}\left(\nu, \pi_x\right)\right)}{\lambda\left(\nu\right)} \\
&= \mathbb{Q}_{T_n}\left(\pi_x\right) \frac{\lambda\left(E_n^x\right)}{\lambda\left(X\right)} \\
&\propto p_n\left(x\right). \qquad (12.10)
\end{aligned}$$

The last equality holds because the area ratios successively cancel each other, and the last proportionality follows from the fact that $\mathbb{Q}_{T_n}$ was defined to implement tournament selection with selection pressure q^{1/T_n}. The ultimate conclusion is that a tree-sampling algorithm with node selection probabilities as given in Equation 12.7 can be used to sample from annealed tournament selection in worst-case logarithmic time.

As a final note on efficiency, notice that sampling in the score tree has worst-case logarithmic time, whereas sampling on the partition tree has average case logarithmic time. Therefore it makes sense to sample annealed proportional selection from the score tree rather than the partition tree. The only additional requirement is that the Taylor coefficients for annealed proportional selection should be propagated up the score tree rather than the partition tree. In this way, regardless of whether tournament or proportional selection is used, the sampling operations of evolutionary annealing require logarithmic time in the worst case.

12.1.4 Implementation

Because evolutionary annealing relies on several data structures, it can be complex to implement. In order to further clarify implementation details and to permit the reproducibility of the experimental results that follow, an open-source implementation was released under the name *pyec* (http://pypi.python.org/pypi/PyEC). This package implements both annealed proportional and tournament selection along with many other popular evolutionary computation methods, including the exact code used to run the experiments described in Section 12.2. This package is intended to encourage further experimentation and evaluation of the evolutionary annealing method beyond the results reported in this dissertation.

Performance statistics for evolutionary annealing were gathered using this implementation in order to demonstrate the actual computational costs of running the algorithm in Table 12.1. These statistics were compiled by averaging results from four runs each of the algorithm using tournament selection on the benchmarks *shekel* and *rastrigin*. Tournament and proportional selection both traverse the score tree when sampling, so the numbers are representative for both selection rules. The columns of Table 12.1 show the average time required for sampling the score tree, for inserting a point into the partition tree, for inserting a point into the ranked score tree, and for the total processing overhead per individual. Each entry shows the average time in milliseconds to process a single individual given a certain number of stored points in the database. The averages are cumulative, so for example the fact that sampling requires 12.9 ms with $100,000$ points in the database means that the average sample time over all $100,000$ individuals was 12.9 ms. As an exception, the total processing time per individual shows the cost per individual averaged over 100 samples. Logarithmic growth in complexity is clear from the table.

Since the implementation details have been fully discussed, the experiments for REA can now be presented.

12.2 Experiments with REA

REA was tested on the twelve benchmarks defined in Table 8.1 using the same methodology as in Chapter 8. The parameters for REA-P and REA-T are the learning rate η and the population size K. Several values for η were tested, shown in Table 12.2 for each benchmark. Preliminary experiments showed that the learning rate influences the performance of REA more than the population size, and thus experiments varying the population size were left for future work. REA-P was not tested in 25 dimensions to conserve computational resources; preliminary experiments showed that REA-T substantially outperformed REA-P in 25 dimensions, just as it does in five and ten dimensions.

Table 12.1: Performance statistics for Evolutionary Annealing on a 2GHz Intel Core 2 Duo processor using the open-source implementation available at http://pypi.python.org/pypi/PyEC. For each number of observed points, the table gives the time in milliseconds for sampling one point, for inserting one point into the partition tree, for inserting one point into the ranked score tree, and for the total processing overhead per function evaluation. Complexity grows logarithmically in the number of points.

points	sample	partition	rank	total
1,000	8.6	18.2	20.6	59.2
5,000	10.5	22.1	24.7	64.5
10,000	11.2	24.1	26.4	68.1
25,000	11.8	27.6	28.2	76.8
50,000	12.4	34.0	30.4	99.2
100,000	12.9	47.3	32.8	113.6

12.2.1 Experimental Results

As in Chapter 8, all algorithms were run on all benchmarks 200 times for each tested parameter setting. These 200 runs are sufficient to guarantee statistical significance on the estimated success rates for each algorithm at the 95% level within ±0.5% [204]. When a single number is shown as the result of an experiment, that number represents the best value achieved on any parameter setting for that algorithm, unless otherwise stated.

The complete experimental results are included in tabular form in Appendix A. Figures 12.2.1 to 12.2.1 show the results for REA-P and REA-T

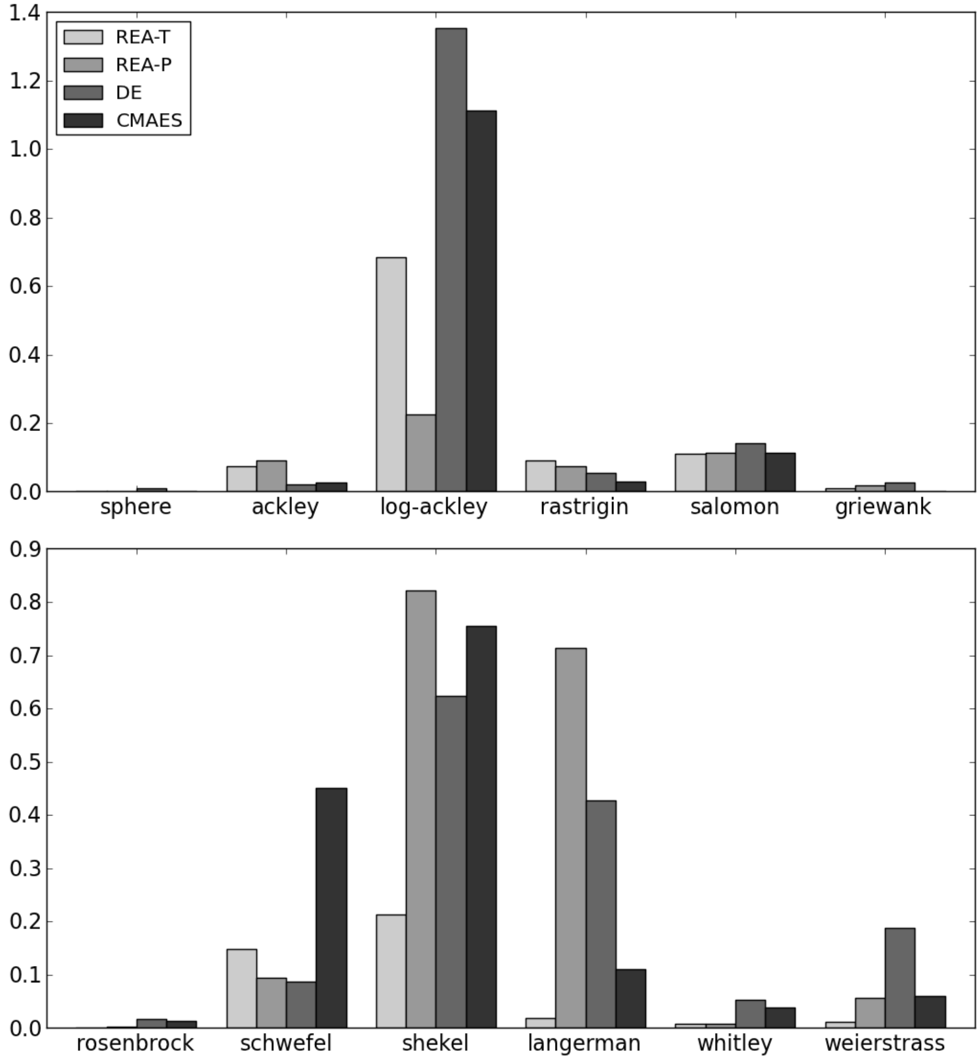


Figure 12.1: Performance of REA, DE, and CMA-ES in five dimensions on the average unweighted error ϕ_1 (scaled), as reported in Figure 8.2 for all optimizers. Lower values are better. All four optimizers are generally comparable on this criterion, with REA-T performing best on *salomon*, *rosenbrock*, *shekel*, *langerman*, *whitley*, and *weierstrass*. REA-T generally has lower error than REA-P, although REA-P also performs well in five dimensions.

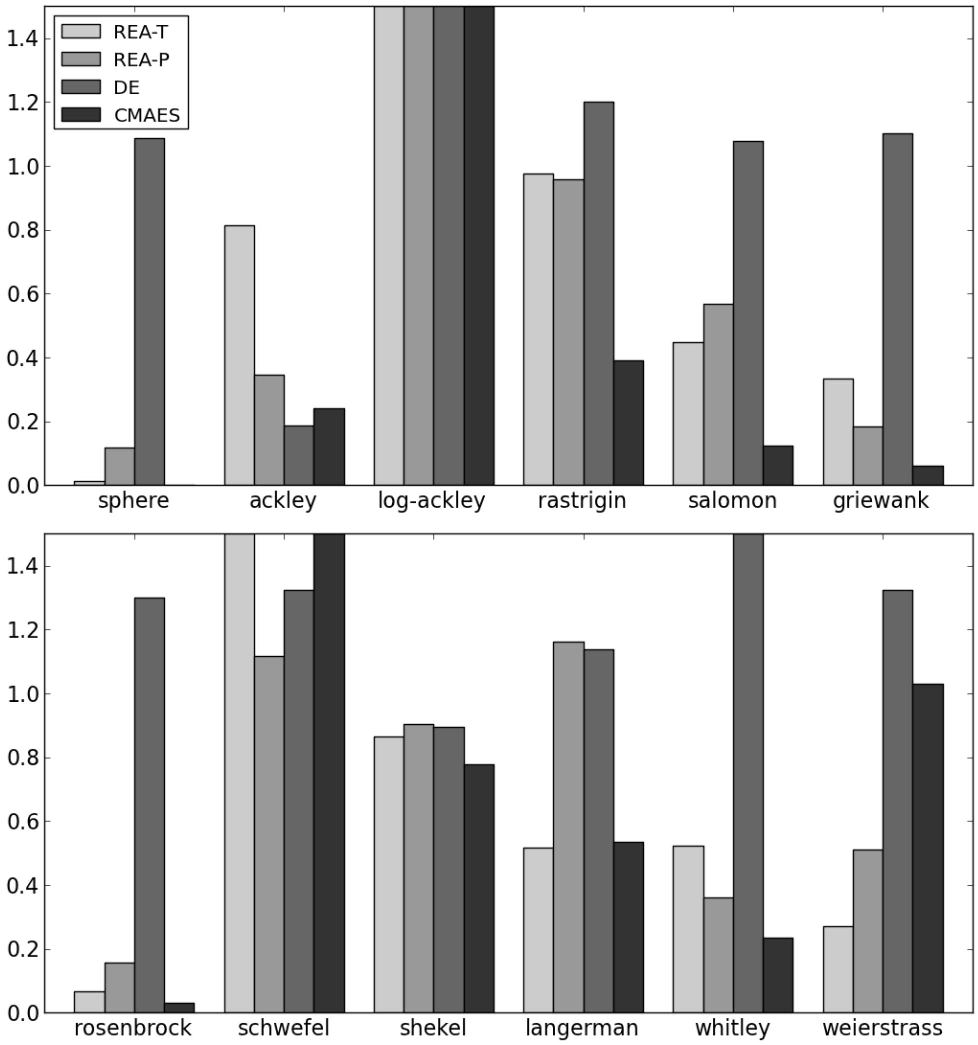


Figure 12.2: Performance of REA, DE, and CMA-ES in five dimensions on the average weighted error ϕ_2 (scaled), as reported in Figure 8.3 for all optimizers. Lower values are better. The criterion ϕ_2 emphasizes early errors, and thus favors faster converging optimizers such as CMA-ES. REA-T converges at the same rate as CMA-ES in several cases, and often makes less early errors than DE. REA-T is the best on *langerman* and *weierstrass* for ϕ_2.

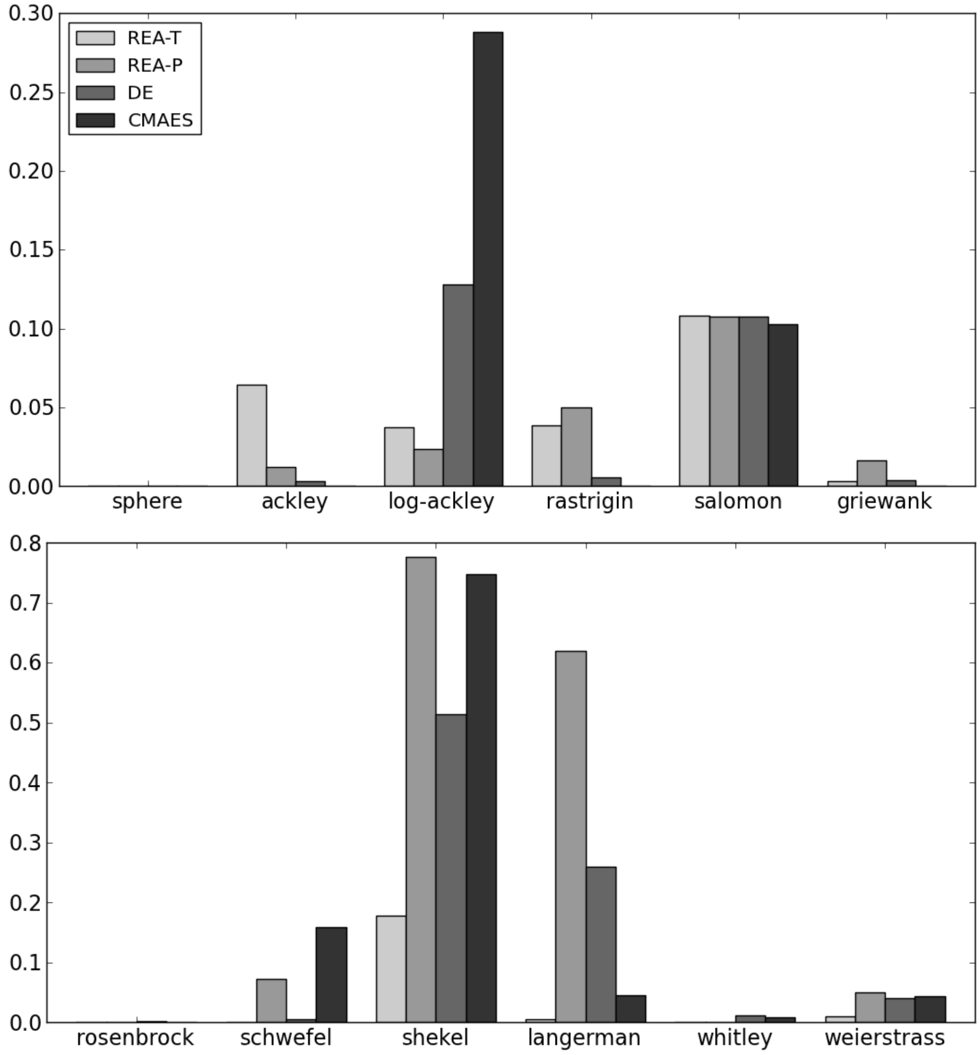


Figure 12.3: Performance of REA, DE, and CMA-ES in five dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled), as reported in Figure 8.4 for all optimizers. Lower values are better. REA-T performs best on *schwefel*, *shekel*, *langerman*, *whitley*, and *weierstrass*, with lower average error at the end of evaluation.

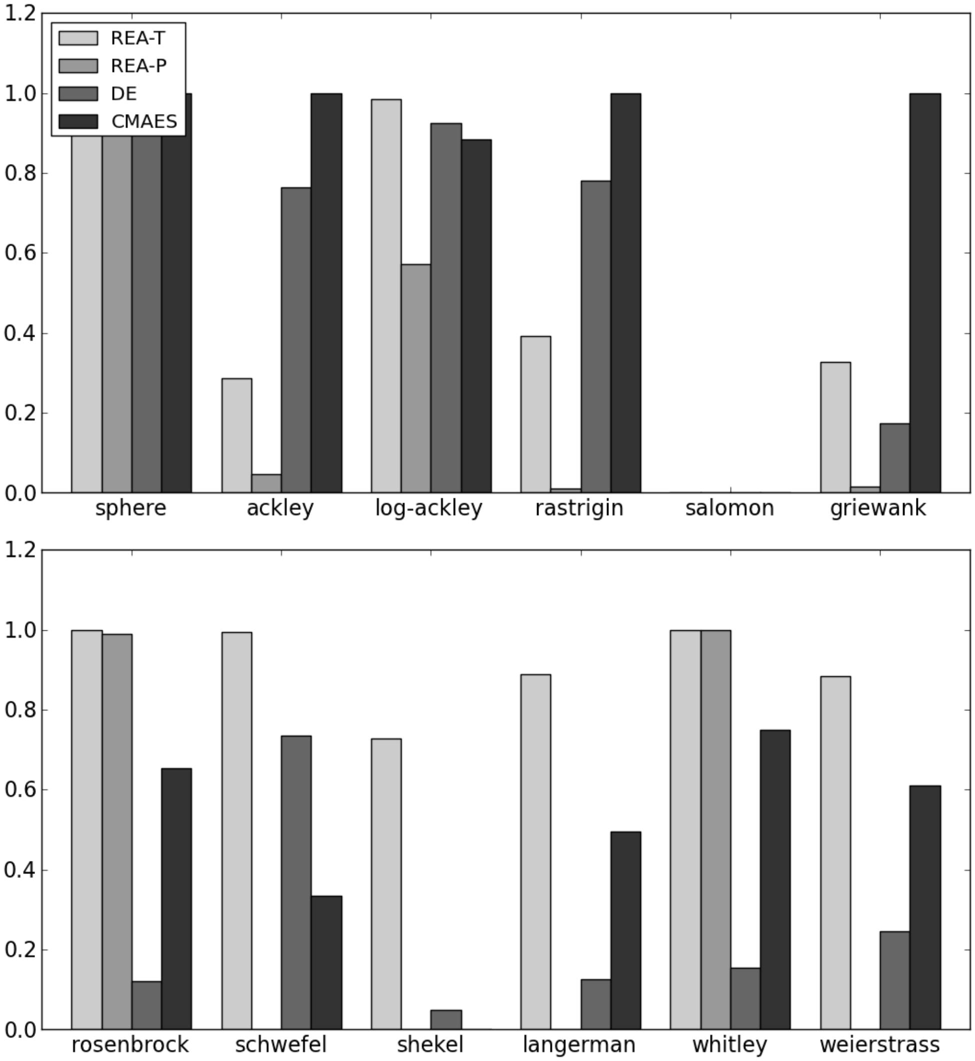


Figure 12.4: Performance of REA, DE, and CMA-ES in five dimensions on the success probability σ_{ϵ}^{N} with $\epsilon = 0.01$ and $N = 250,000$, as reported in Figure 8.7 for all optimizers. Higher values are better. REA-T generally performs best on the irregular problems, such as *shekel*, *langerman*, and *whitley*.

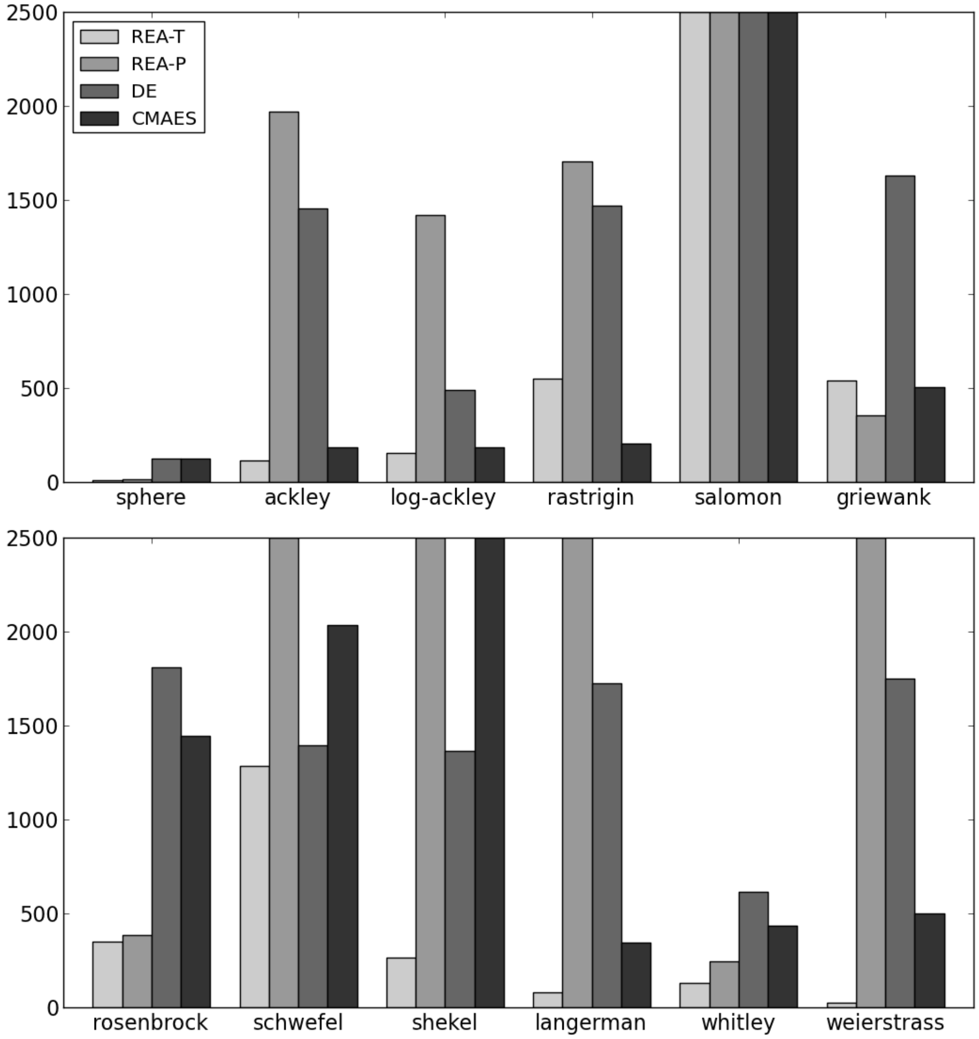


Figure 12.5: Performance of REA, DE, and CMA-ES in five dimensions on the average hitting time $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.01$ and $N = 250,000$, as reported in Figure 8.6 for all optimizers. Lower values are better; 2500 is maximum value. REA-T converges at about the same rate as CMA-ES when successful, suggesting that its performance could be boosted by restarting. In general, REA-T is more accurate than CMA-ES.

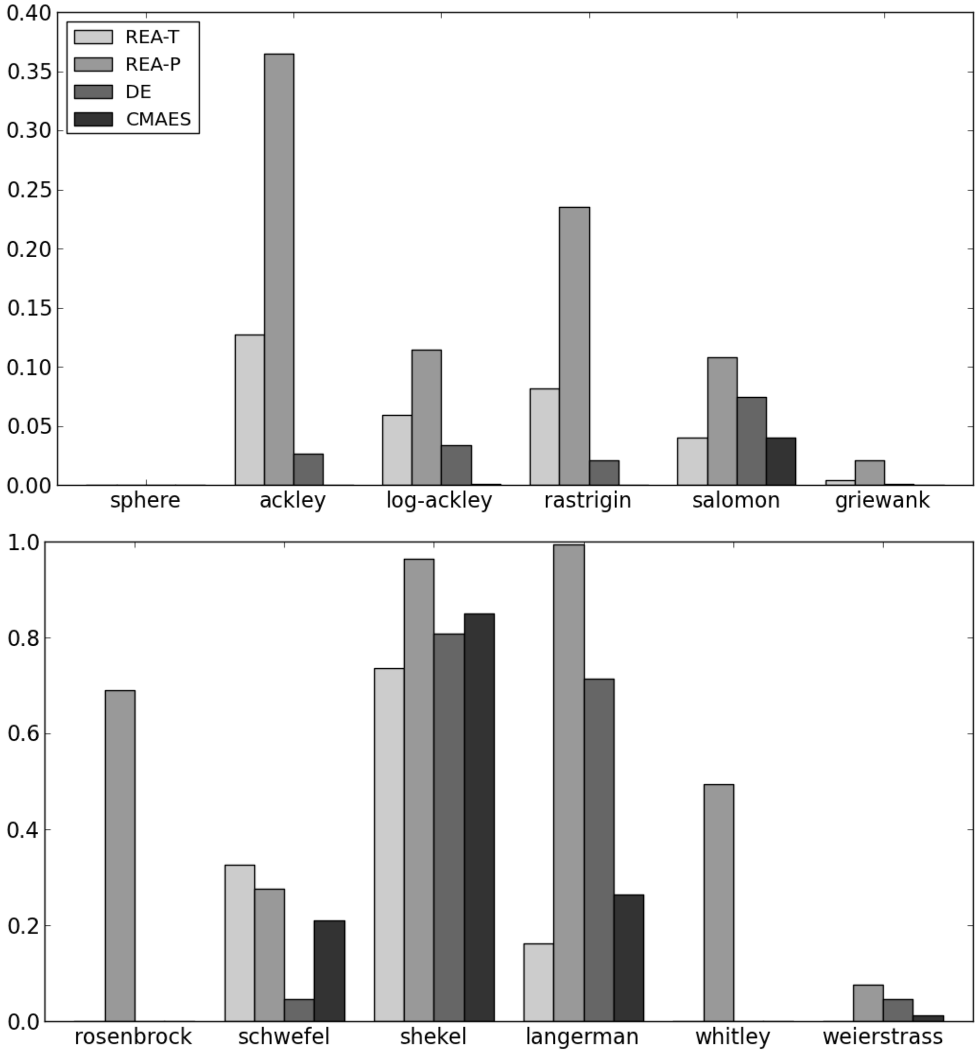


Figure 12.6: Performance of REA, DE, and CMA-ES in 10 dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled). The performance of REA-P degrades as the dimension increases. REA-T performs best on several problems, including *salomon*, *shekel*, *langerman*, and *weierstrass*.

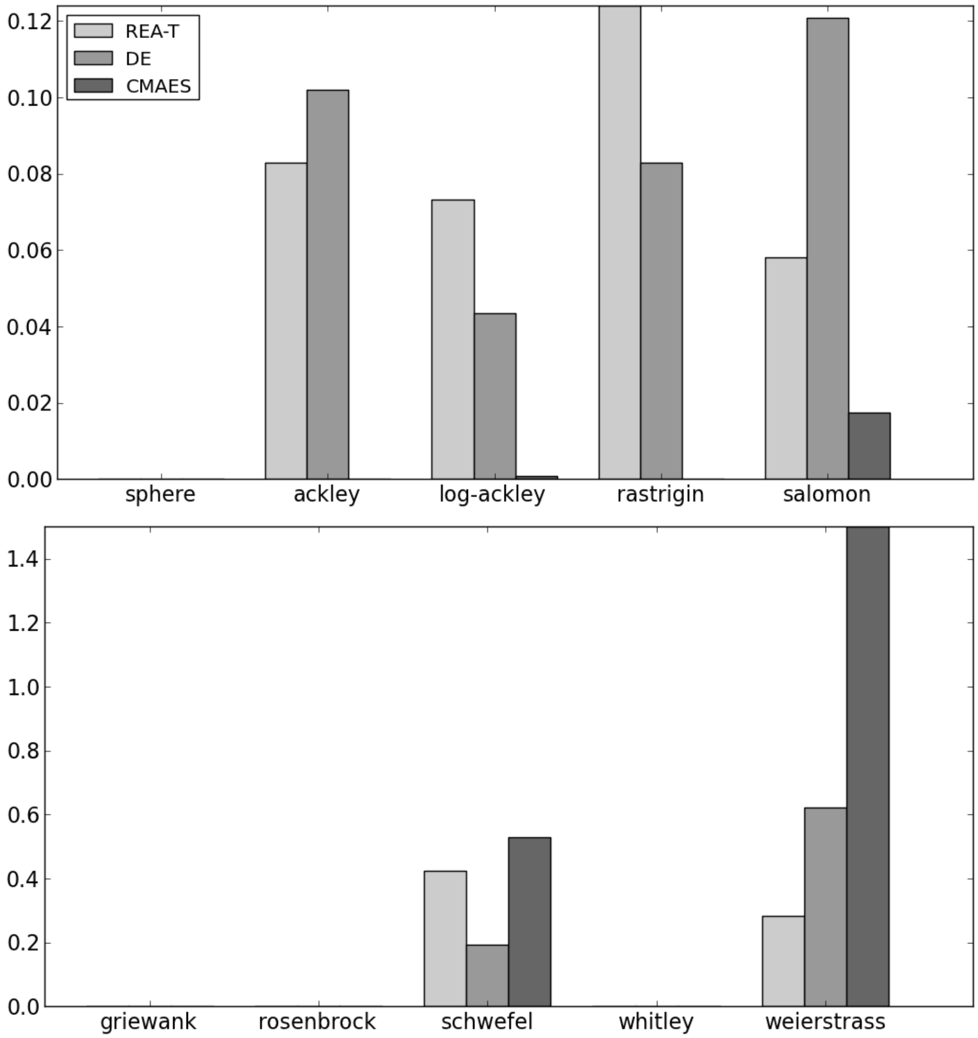


Figure 12.7: Performance of REA, DE, and CMA-ES in 25 dimensions on the average final error $\zeta_{T_{250,000}}$ (scaled). REA-P performed substantially worse and was omitted. All algorithms perform relatively well on *sphere*, *griewank*, *rosenbrock*, and *whitley*. REA-T performs best among these three methods on *weierstrass*.

Table 12.2: Learning rates η for REA-P and REA-T tested in the experiments. Lower values yield higher success probability at the cost of slower convergence.

	REA-P		REA-T		
Benchmark	$d = 5$	$d = 10$	$d = 5$	$d = 10$	$d = 25$
sphere	10	1, 10	10	1, 10	0.1, 1,10
ackley	0.25	0.25, 1	0.25	0.25, 1	0.05, 0.25, 1
log-ackley	0.25	0.25, 1	0.05, 0.25	0.25, 1	0.05, .25, 1
whitley	0.1	0.25, 1	0.05, 0.25	0.25, 1	0.05, 0.25, 1
shekel	0.1, 0.25	0.1, 1	0.1, 0.5, 1.0, 5.0	0.1, 1	–
rosenbrock	1	1, 5	5	1, 5	0.1, 1,5
rastrigin	0.01, 0.1	0.035, 1	0.01, 0.035, 0.050, 0.075	0.035, 1	0.01, 0.035, 1
salomon	2	1, 2	2	1, 2	0.1, 1, 2
langerman	0.1, 0.5	0.25, 1	0.1, 0.5, 1.0, 5.0	0.25, 1	–
schwefel	0.015	0.001, 0.01	0.001	0.001, 0.01	0.0001, 0.001, 0.01
griewank	1, 10	0.1, 1	0.025, 0.1, 0.25, 0.5	0.1, 1	0.01, 0.1, 1
weierstrass	5	1, 5	5	1, 5	0,1, 1, 5

in five dimensions on various performance criteria from Section 7.1, with DE and CMA-ES included for comparison. Results for REA-T in 10 and 25 dimensions on ζ_T are shown in Figure 12.2.1 and 12.2.1.

In short, REA-T, DE, and CMA-ES are the most effective optimizers on this set of benchmarks, with restarted optimizers excluded. REA-T is more effective on problems that are asymmetric, non-separable, and multimodal such as *shekel*, *langerman*, and *whitley*. In Section 12.3, this fact will be discussed in terms of alignment with a particular function prior. DE outperforms REA-T on some but not all radially symmetric problems such as *rastrigin*, *salomon*, and *griewank*. CMA-ES performs particularly well on *rastrigin* and *griewank*. Comparing the two versions of REA, REA-P performs well, but fails to refine solutions near global and local optima. Its performance also degrades in higher dimensions. In contrast, REA-T attains precisely refined solutions, most often at the global optimum, and is therefore the stronger method on these benchmarks.

More specifically, in five dimensions, the results show that REA-P and REA-T are effective at locating the global optima of complex fitness functions. REA-P is successful on most problems at the 0.1 success level, with notable exceptions for *rastrigin* and *schwefel*. For *schwefel*, REA-P actually located the region of the true global optimum on most trials, but was unable to refine these solutions further. For comparison, the failures of CMA-ES and PSO on this benchmark were over an order of magnitude worse and were not in the correct region of the search space. On *rastrigin*, it was not possible to configure REA-P to succeed predictably. The algorithm may succeed at a lower learning rate (e.g. $\eta = 0.001$) with more function evaluations, but an even lower learning rate would further slow down the refinement of the solution.

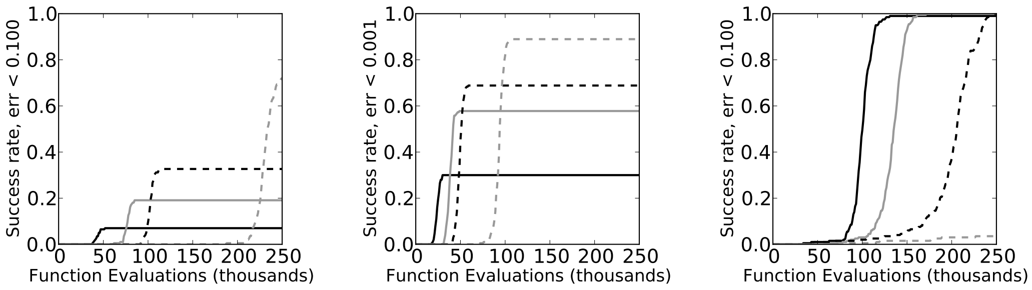
By contrast, REA-T is very effective at refining points around the optima. In most cases where REA-T came within 0.1 of the optima, it also managed to attain machine-level precision. The exceptions to this statement primarily involved local optima with fitness values close to those of the true optimum (i.e. *salomon*, *langerman* and *griewank*). In the case of *rastrigin*, tournament selection even helped REA-T escape local optima in several cases, so that it attained the true global optimum more often than REA-P.

In higher dimensions, all of the algorithms had trouble attaining the global optimum. However, a review of the errors in Table A.41 shows that REA-T was competitive with the others. In preliminary trials, REA-P failed on *whitley* and *rosenbrock* as a consequence of numeric issues. In both of these problems, the region of the search space containing reasonable fitness values (e.g. $f(x) < 100$) is small relative to the overall area, and in higher dimensions this region becomes exponentially smaller. Annealed proportional selection overflows on large fitness values (Equation 11.4) and must therefore be capped, so the probability that REA-P selects any particular point is effectively constant. This problem can be overcome by using a very small learning rate, but then REA-P would not be able to converge once the feasible region is attained. Because annealed tournament selection is only sensitive to the fitness rank of points, REA-T does not suffer from numeric issues and continues to perform relatively well on *whitley* and *rosenbrock* even in higher dimensions. It is possible that with lower learning rates, REA-T could perform even better in 25 dimensions.

Figure 12.8 shows the progression of the success probability and Figure 12.9 the magnitude of the error as a function of the number of evaluations for REA-T with different learning rates on selected benchmarks. As the learning rate is decreased, REA-T converges slower and succeeds more often. Thus there is a trade-off between the number of evaluations and solution quality. A higher learning rate can be used to reduce the number of evaluations, but at the cost of reducing the probability of success. Notice that the shape of the graph remains remarkably constant in Figure 12.8 while the learning rate changes, suggesting that the success probability changes smoothly and predictably as a function of the learning rate and the number of evaluations.

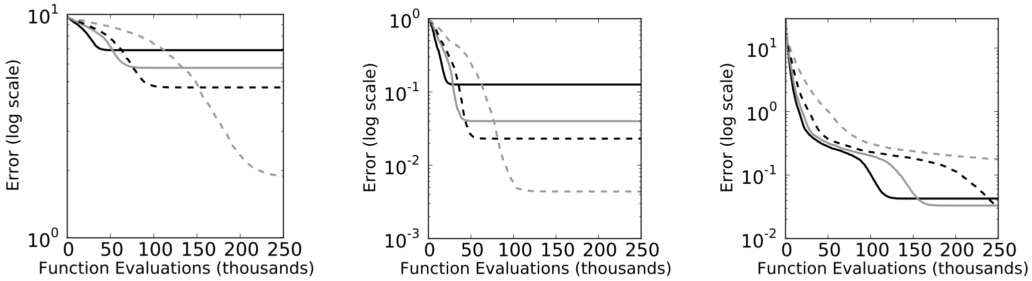
12.2.2 Analysis of REA Results

The experimental results in Section 12.2 favor evolutionary annealing, especially with annealed tournament selection. There are some generalizations that may be drawn from the results. First, REA-T is generally better than REA-P for optimization and is thus the preferred implementation for Euclidean space. Second, REA is most successful relative to other algorithms on problems that do not possess an easily identifiable structure, such



(a) shekel (b) langerman (c) griewank

Figure 12.8: Success probabilities for REA-T on selected benchmarks in five dimensions for four different learning rates. Decreasing the learning rate improves the success probability overall but requires more fitness evaluations.



(a) shekel (b) langerman (c) griewank

Figure 12.9: Average error rates for REA-T on selected benchmarks in five dimensions for four different learning rates. The black solid line is the average error for the largest learning rate in Table 12.2; the grey solid line is the second largest; the black dotted line is the third largest, and the grey dotted line is the smallest learning rate. Decreasing the learning rate thus reduces error overall at the cost of increased error in early generations.

as *langerman* and especially *shekel*. The reason is that REA does not assume a particular problem structure in its definition. This observation is discussed further in Section 12.3. In structured domains, such as *sphere*, REA may use more function evaluations than would otherwise be necessary to eliminate the possibility that the current best solution is a local optimum. However, in unstructured environments, these extra function evaluations help REA avoid becoming trapped in local optima.

Among the non-restarted algorithms, REA-T is most comparable to DE in terms of optimization quality. DE is an elegant and simple algorithm and is consequently more computationally efficient than REA-T, performing up to two orders of magnitude faster in terms of per-generation overhead. However, in real-world problems, the computation of fitness values typically far outweighs the cost of algorithmic overhead. The overhead of REA is generally unrelated to the fitness function being optimized, so in domains where the fitness takes a long time to compute, the use of REA will not add substantially to the overall computation time.

Also, the results on the benchmarks suggest that DE and REA-T are complementary, with REA-T being preferable on highly unstructured problems, and DE performing better on problems with some degree of symmetry around the optimum. In practice, there are many real-world problems both with and without symmetry. If the degree of structure is not known, and fitness can be calculated quickly, a reasonable approach is to test DE first and use REA-T if DE fails.

All of the restarted algorithms (NM-R, GSS-R, and CMA-ES-R) generally performed as well or better than REA-T on most benchmarks, with the notable exception of *langerman*. Restarting after convergence is a form a boot-strapping that can augment the probability of success. For example, if an algorithm has a 5% chance of success, but converges after $1,000$ evaluations, then by running the algorithm 100 times, that 5% success rate can be boosted to 99.4%. To benefit from numerous restarts, and algorithm must obtain a positive success rate quickly. For REA, if the learning rate η is set at a high level (e.g. > 1), then REA-T will converge quickly. If this convergence can be measured, then REA-T can be restarted to boost its success rate as well. Such an extension is an interesting direction for future work.

In contrast to the other successful optimizers, evolutionary annealing is well-defined in any suitable measure space. Thus evolutionary annealing can be used to search for neural networks, game strategies, Bayesian network structure and many other problem domains where it is unclear how DE, CMA-ES, NM, GSS, or PSO might be applied. In fact, preliminary experiments have been performed in all these problem domains with promising results.

The benchmark set also shows that REA performs well on problems to which it should not be particularly well-suited, at least while using Gaussian variation. For instance, separable problems such as *schwefel* and *weierstrass* can be more efficiently solved by searching in only one dimension. The optimizer rGA succeeds on *schwefel* by using recombination to cross-pollinate correct components, and DE succeeds by sharing component-level information among the different members of its population through its unique crossover mechanism. In contrast, REA must learn each component separately. While this aspect of REA could be improved for *schwefel* by implementing a mutation distribution that employs crossover, it is nonetheless promising that REA is able to learn the correct value for all components independently without using excessively more function evaluations than the other algorithms.

Given that REA-T is designed to search a space exhaustively for the global optimum, it might be expected to perform worse than more greedy algorithms in higher dimensional spaces. The results show that the opposite is true: REA-T still performs among the best algorithms tested even in 25 dimensions. One reason is the addition of the decay factor $n^{-\frac{1}{2}}$; without this decay factor, REA-T failed to find good solutions in 25 dimensions. To see why, consider that in d dimensions, 2^d evaluations must be performed in order to cut the average side length of a partition region E_n^a in half. Thus the variance $\sigma_n(a)$ reduces exponentially slowly in higher dimension. The decay factor forces evolutionary annealing to focus only on the most promising solutions. In this way, evolutionary annealing can obtain good solutions in reasonable time for higher dimensions at the cost of global optimality.

In Chapter 13, evolutionary annealing will be applied to neural networks, and that application will benefit from the results of the experiments in this chapter. The purpose of defining evolutionary annealing at the chosen level of abstraction is to provide a means for developing new algorithms to search in complex spaces without having to reinvent the underlying evolutionary apparatus from whole cloth. Evolutionary annealing provides convergence guarantees as well as heuristics for setting learning parameters for a wide variety of search domains.

More work remains to be done to establish the rate of convergence for evolutionary annealing beyond the heuristics provided in Section 11.3.1. For example, maximum likelihood estimates of mixture distributions with increasing mixing points are known to approximate continuous distributions at a relatively fast rate of $C\left(\frac{\log n}{n}\right)^{0.25}$ [72]. The distributions employed in evolutionary annealing are not the same, but similar performance may be hoped for on continuous fitness functions. Also, theoretical work needs to be done to find sufficient conditions on the cooling schedule and mutation distributions to make an evolutionary annealing algorithm exhaustive.

Ultimately, the success of evolutionary annealing must be determined by experimentation in real-world applications. It is difficult to predict in advance whether evolutionary annealing will be successful in such applications, but the results on benchmarks make it clear that evolutionary annealing is worthy of consideration as a method for global optimization in general-purpose domains. The next section analyzes the types of problems on which evolutionary annealing should perform best.

12.3 Problem Alignment

REA was presented as a general optimization technique for arbitrary problems in Euclidean space. However, in line with the discussion in Section 10.2, it may be expected that there are function priors on which REA performs better than other optimizers. In fact, the experimental results just presented provide evidence for this sort of alignment. REA outperforms other optimizers on irregular, multimodal objectives such as *langerman*, *shekel*, and *whitley*. So what is the natural function prior corresponding to REA, i.e., what sorts of problems play to REA's strengths?

REA is a model-building optimizer. At each time step, REA builds a probability distribution defined by

$$\mathbb{P}(dx) = \xi_n \sum_{a \in A_n} \frac{\lambda(E_n^a)}{\sigma_n(a)\,(2\pi)^{d/2}} \exp\left(-\frac{f(a)}{T_n} - \frac{|x-a|^2}{2\sigma_n(a)^2}\right)\lambda(dx). \tag{12.11}$$

This distribution is a mixture of Gaussians, where the number of mixing points increases with each time step and the variance is a decreasing function of the number of mixing points. In light of Section 10.4.5, REA might be expected to perform best when the conditional expectation under the prior has a shape similar to the λ-density in Equation 12.11.

In the initial generations, the distribution in Equation 12.11 has relatively few large modes distributed broadly through the search space. As the number of function evaluations increases, the modes become smaller, but the points explored become closer to each other, since REA emphasizes exploration within the modes it has already discovered. Thus REA expects to discover secondary modes distributed across the modes already known. Extrapolating out to infinite time, the well-aligned function prior should prefer functions with a fractal structure that results from the composition of many Gaussian modes overlaid in tight clusters and distributed sparsely throughout the search space. The location of large modes might appear as though drawn from a Dirichlet prior at various levels of refinement. With high probability, the existing modes would be maintained, and with low probability a new mode would be sampled.

As an example of this kind of prior, consider the following iterative sampling scheme for functions over $\mathbb{R}^1$ on the interval $[-10, 10]$. First, choose 10 points $x_1^0, \ldots, x_{10}^0$ distributed uniformly over the interval. Assign a value $F(x_1^0)$ uniformly at random on $[0, 1]$. Then for n greater than zero, sample $x_1^n, \ldots, x_{10}^n$ from a mixture distribution with density

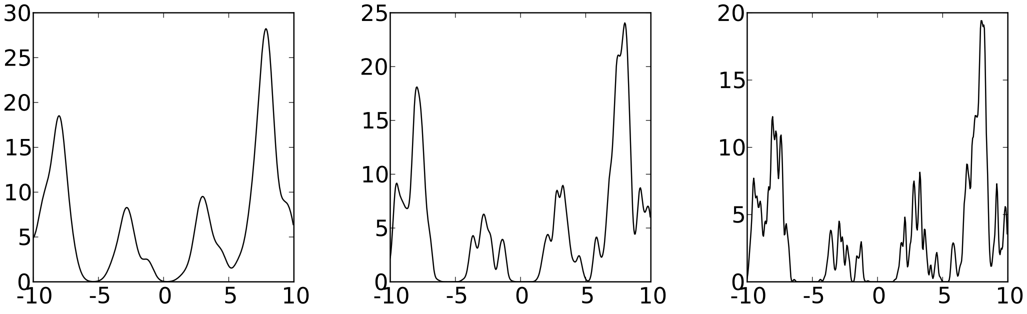
$$G_n(x) = \xi_n \sum_{a \in \bigcup_{i=1}^n \bigcup_j \{x_j^i\}} \frac{1}{1.035^{-n}\sqrt{2\pi}} \exp\left(F(a) - \frac{|x-a|^2}{2 \times 1.035^{-2n}}\right), \tag{12.12}$$

where ξ_n is a normalizing factor. Notice the similarity to Equation 12.11, except for the area-sensitive variance. Consider each G_n as a random objective, and notice that the conditional expectation of G_n based on the first $10m$ points obeys the equation

$$\mathbb{E}\left[G_n \mid F(x_j^i),\, i \leq m\right] = C \sum_{a \in \bigcup_{i=1}^{m} \bigcup_j \{x_j^i\}} \frac{1}{1.035^{-n}\sqrt{2\pi}} \exp\left(F(a) - \frac{|x-a|^2}{2 \times 1.035^{-2n}}\right), \tag{12.13}$$

where C is a constant reflecting the normalizing factor and the expectation of the remaining $10(n-m)$ terms from the sum in G_n. Annealed selection (proportional or tournament) is unchanged by the addition of a constant multiplier, since such terms are normalized out. Thus the annealed selection rules are approximately martingales on the sequence of priors given by G_n, supposing that the information contained in $F(x_j^i)$ for $i \leq m$ is the same or similar to the information in $G_1, \ldots, G_m$. Loosely, then, it seems that evolutionary annealing implements the information maximizing strategy in Section 10.4.4 for the prior $G_\infty = \lim_n G_n$, with the final step of that strategy being unnecessary, since the expected minimum is one of the x_j^i. Figure 12.3 shows an objective sampled from the G_n at G_{25}, G_{50}, G_{75} as an example of the sorts of objectives that might be generated by this procedure. These images show the type of function on which REA should perform well. Note that if the fixed variance decay in G_n were replaced with REA's variance-sensitive decay, the generated objectives would probably be slightly smoother, especially in regions where the x_j^i are more sparse. Examining the progression of G_n reveals the fractal nature of such a function prior.

Examining the heat maps in Figure 8.1, it may be seen that the procedure from the last paragraph most accurately describes the benchmarks *langerman*, *shekel*, and to some extent *whitley*. The benchmark *langerman* in particular fits the description. It has four large modes, two of which overlap, with rugged and detailed volcano-like structures at the top of each mode. Thus the intuition about what kinds of priors REA might prefer is confirmed by the experiments, and REA should be considered as a good alternative for optimizing multimodal functions with generally irregular structure.



(a) G_{25 (b) G_{50} (c) G_{75}

Figure 12.10: Examples of priors drawn from G_{25}, G_{50}, and G_{75} using Equation 12.12. The random objective G_∞ may be well aligned with REA, supposing a fixed, decaying variance was used in place of REA's area-sensitive variance decay. An area-sensitive decay would be more smooth, particularly in sparse regions. The fractal structure of G_n is clearly evident in these images. This function roughly resembles *langerman*, on which REA-T performs best among all optimizers tested, suggesting that REA is well aligned with this random objective.

12.4 Conclusion

In this chapter, REA was applied to optimization in finite-dimensional Euclidean space. Details were offered for an efficient implementation of annealed selection. Experiments on the benchmarks from Chapter 8 showed that REA performs well in comparison to other optimization methods, particularly with annealed tournament selection. The performance profile of REA was found to be complementary to the performance of the other optimizers that were tested. This complementarity results from the alignment of REA to function priors that generate irregular objectives with good fitness distributed sparsely among different modes at several fractal levels.

Euclidean space is a common target for optimization, but evolutionary annealing can be applied to other spaces as well. The next chapter studies evolutionary annealing as a tool for optimizing neural networks.

Chapter 13

Neuroannealing

In the previous chapter, evolutionary annealing was applied to bounded subsets of Euclidean space to demonstrate that an information maximizing approach to optimization is both feasible and effective. An important feature of evolutionary annealing is that it can be applied to any measurable space. In this chapter, evolutionary annealing is employed to search a space of recurrent artificial neural networks; this approach to learning neural networks will be termed *neuroannealing*. Neuroannealing is compared to a successful neuroevolution method, NEAT, and is shown to perform better on certain kinds of problems, in particular, those that require large neural networks with deep structure. As will be discussed in this chapter, neuroannealing is able to discover complex solutions because it retains all previously tested solutions, allowing it pass through regions with lower objective values in order to reach the solutions.

13.1 Evolving Recurrent Neural Networks

Recurrent neural networks (RNNs) are a flexible class of parameterized nonlinear dynamic functions. In a supervised setting, the dynamics of an RNN can be learned using techniques such as Backpropagation Through Time [175]. However, RNNs are often applied to control tasks, where a supervised learning signal is not generally available. In such cases, the two most prominent approaches for training neural controllers are reinforcement learning [176, 200, 209, 214] and evolutionary computation, termed *neuroevolution* [66, 80, 81, 99, 138, 196, 222]. Neuroevolution, especially advanced neuroevolution methods like NEAT [196], has been shown to be more effective in certain control tasks. Evolutionary annealing was compared to several competing optimizers in Chapter 12. In this chapter, neuroannealing will be compared experimentally to NEAT in order to demonstrate the benefits of the annealed selection for learning neural networks.

13.1.1 RNN Basics

An RNN consists of a set of artificial *neurons*, or *nodes*, connected by artificial *synapses*, or *links*, with a signal-modulating *weight*. A subset of the nodes, termed the *input nodes*, are used as sensors to observe external state. A disjoint subset, the *output nodes*, are treated as the network's output signal. The remaining nodes are referred to as *hidden nodes*. In addition, most RNNs use a *bias* on each node to predispose the neuron to be more or less easily activated. Computation in an RNN proceeds by propagating an input signal through the synapses until equilibrium, and then measuring the *activation* or *excitation* of the output nodes. An RNN is characterized by the fact that the network graph, formed by taking the neurons as nodes and the synapses as edges, may contain loops. A neural network without loops is termed a *feedforward neural network*.

An RNN is determined by its connectivity and its weights. The *network topology* refers to the particular pattern of connectivity within a network. It is not the same as a topology of a space, although the two are distantly related. Two disconnected neurons may be described as being connected with a zero weights, and thus one mathematical representation of an RNN uses a pair of weight matrices, one for connections to the input (the input weights), and one for connections among the hidden and output nodes (the hidden weights). For an RNN with N inputs, H hidden nodes and M outputs, the input weight matrix I is an $(H + M) \times N$ matrix, the hidden weight matrix W is an $(H+M)\times(H+M)$ matrix, the bias b is an M-dimensional vector. Collectively, I, W and b constitute the parameters of an RNN.

The state of the RNN is a vector in $\mathbb{R}^{H+M}$ that assigns a real number to each output and hidden node. If x_n is the state of a network, then given an input u_{n+1}, the next state x_{n+1} is computed as

$$x_{n+1} = \sigma\left(I u_{n+1} + W x_n + b\right), \tag{13.1}$$

where σ is a nonlinear *activation function*, often called a *squashing function* because it is usually intended to compress the neuron state within a small finite range. Typical activation functions are the hyperbolic tangent, $\sigma(x) = \tanh(x)$, and the logistic function, $\sigma(x) = (1 + \exp(-x))^{-1}$. The hyperbolic tangent compresses activation values into $[-1, 1]$, and the logistic compresses them to $[0, 1]$. In this chapter, neuroannealing uses the hyperbolic tangent, and NEAT uses the logistic function. There is no significant difference between the two in terms of computing power.

A feedforward neural network with enough nodes and sufficiently precise weights can approximate any integrable real function [49]. Discrete-time RNNs are strictly more powerful than feedforward networks. In terms of computational theory, every binary language is decidable by some RNN with real weights, meaning that RNNs are capable of performing tasks that a Turing Machine cannot [185]. This result remains true even if the RNN is only run for a finite number of steps [35]. With rational weights, RNNs are at least as powerful as Turing Machines [185].

As dynamical systems, most RNNs are Lyapunov-stable and converge to equilibrium exponentially fast [19], meaning that their neural activations tend towards a static equilibrium in very few steps when the inputs are fixed. Thus an RNN with random weights and no inputs cannot generally compute an arbitrary time sequence. However, a specially constructed RNN can generate limit cycles [174]. Such limit cycles are induced by a chain of neurons arranged in a singly-connected loop; in such a chain, the activation is passed along each neuron, generating a time-varying source that does not depend on the inputs. This fact is utilized in developing the neuroannealing approach.

13.1.2 Neuroevolution

The term *neuroevolution* describes the process of applying evolutionary algorithms to search a space of neural networks to find a network that optimizes some fitness criterion. In this approach, the parameters of a neural network are encoded inside of one or more artificial genes, which are then selected and mutated to form new networks. Early work focused on networks with fixed size and topology [44, 66, 150, 213, 222]. With this assumption, it is straightforward to encode a neural network as a vector in $\mathbb{R}^C$ where C is the number of connections in the network.

Subsequent work resulted in methods for evolving networks one neuron at a time, averaging over the performance of different networks to estimate the value of particular parameters. Such methods include SANE [138], ESP [81], and CoSyNE [80]. All of these methods use a fixed number of hidden nodes.

In a different vein, NeuroEvolution of Augmenting Topologies (NEAT) was introduced as a neuroevolution algorithm that seeks to produce only those hidden nodes that improve the overall fitness of a recurrent neural network [195, 196]. NEAT has been widely applied to several experimental settings with success [128, 130]. Later in this chapter, neuroannealing will be compared experimentally with NEAT, and so some discussion of the algorithmic details is necessary.

NEAT is initialized with a population of networks that contain no hidden nodes and no recurrent links. These networks consist only of input nodes directly connected to the output nodes. In successive generations, NEAT uses proportional selection to choose a pair of network parents. It then applies crossover (either intermediate or multipoint crossover). After crossover, a network may undergo one or more modifications, either adding a node, adding a link, or mutating an existing weight. Weight mutation applies a relatively large Gaussian ($\sigma \approx 2$) to the current weight. New links are added with a small random weight. When a new node is added, it replaces an existing link between any two connected nodes. In this case, two new connections are also added. One connection is added from the source of the link to the new node with a weight of 1.0. A second connection is added from the new node to the target of the original link, copying the original weight. The general purpose of these details is to preserve network function as much as possible. If a structural mutation substantially impairs the performance of a network, then the new mutation will be immediately ejected from the population. Adding nodes and links using the method above increases the chance that the new network will survive.

NEAT has several additional features that improve its performance. *Speciation* segregates the population of networks into subgroups based on the similarity of topology and weights, and crossover is restricted so that both parents are usually but not always drawn from the same species. The measure of similarity can be tightened or relaxed. The use of species in NEAT preserves suboptimal solution candidates that are different from existing solutions during reproduction, promoting more thorough exploration of network topologies. Species are allowed to go extinct after a fixed number of generations with no improvement. Additionally, NEAT marks each new structural feature (i.e. a new node or connection) with a unique identifier, so that when crossover is applied to networks with different structures, the shared structures can be properly aligned. NEAT also uses *elitism*, retaining the best member of the last population, except when the species containing the best member of the population goes extinct. There are many other details required to describe NEAT fully, and they make a difference in how well NEAT performs. It is thus difficult to give complete mathematical account of NEAT's behavior. The source code for NEAT is publicly available, and this code was used to test NEAT in the experiments below.

13.1.3 Evaluating the Performance of NEAT

Neuroevolution in general has been shown to perform well in control tasks, such as controlling a finless rocket in flight [79] or generating a gait for a multi-legged robot [203]. Experiments by Stanley showed that NEAT performs well in a number of domains, including pole-balancing, board games, obstacle avoidance in driving simulations, and control of virtual robots [197]. In general, NEAT quickly locates small and efficient recurrent networks that solve a task.

NEAT does not always perform well, however. The failure modes of NEAT were studied by Kohl [110], who found that NEAT's performance tends to degrade with the complexity of the problem, as determined by the total variation of the problem. A neural network defines a map between input states and output states. Such a map will be termed a *state-action map*. Neuroevolution searches through the space of neural networks as a tractable proxy for the space of state-action maps. Let Π be any partition of the search space consisting of hyperrectangles, and suppose that action space is metric. The *variation* of a state-action map on a hyperrectangle H is the largest distance between the action values at any two corners of H. The *total variation* of a state-action map over the partition Π is the sum of the variation of the map on each hyperrectangle in Π. The total variation of a state-action map is the supremum over all partitions consisting of hyperrectangles. The total variation of a problem is the infimum of the total variation of state-action maps that solve it.

Kohl showed that the performance of NEAT degrades as the total variation of the problem increases, a property that he termed *fracture* [110]. Kohl exhibited several problems with fractured state spaces, such as recognizing concentric spirals, implementing a multiplexer for address-based lookup, and robotic keepaway soccer. He also proposed a solution using radial basis function nodes that improved NEAT's performance on these domains. As Kohl observed, when NEAT does succeed in fractured domains, the successful networks tend to be larger, allowing them to encode higher complexity that reflects the fractured problem domain. Kohl's solution works because the mix of radial basis functions with sigmoidal nodes allows compact networks to exhibit more complex behavior. Kohl also experimented with cascaded networks in which the existing weights of the network are frozen and new sigmoidal nodes are added, which was also successful. In contrast, neuroannealing allows all weights in a network to change throughout training.

13.1.4 Experimental Hypothesis

In contrast to Kohl's approach, neuroannealing does not add radial basis functions and does not freeze weights, but is still able to find more complex solutions to fractured problems. It is able to do so because it retains information from all prior evaluations in order to generate new candidate solutions.

In order to move from a simple network with reasonably good performance on the objective to a complex network with better performance, an optimization method must either make all structural changes to the network in one step, or else it must make a series of incremental changes, each of which may degrade the objective value of the network. As a population-Markov optimizer, NEAT discards previously evaluated networks that fail to improve fitness. The speciation mechanism used by NEAT preserves novel structure for a period of time, but any network that does not improve performance is eventually eliminated. The probability that the required intermediate steps are preserved in the population under NEAT therefore decreases exponentially with the number of steps required.

In neuroannealing, the intermediate solutions remain in the pool of previously observed networks, and thus it is possible to discover more complex networks that achieve higher fitness. As a result, neuroannealing should outperform NEAT on fractured problems. However, the fact that more exploration is performed around suboptimal points imposes a cost in terms of the number of evaluations. When NEAT succeeds, it should succeed faster than neuroannealing.

This hypothesis will be tested experimentally on concentric spirals, multiplexers, double pole-balancing, and automated currency trading. First, the application of evolutionary annealing to the space of neural networks is described.

13.2 Evolutionary Annealing for Neural Networks

In order to apply evolutionary annealing to the space of neural networks, three components must be defined: (1) a base measure over neural networks, (2) an algorithm for partitioning sets of neural networks, and (3) a sequence of mutation distributions likely to improve the objective value of a network. This section proposes a particular approach to defining these components that is collectively termed *neuroannealing.* First, the concept of a layer of nodes is introduced as a building block for RNNs, and then each of the three components are described in turn.

13.2.1 Layered RNNs

Neuroannealing searches the space of RNNs for the optimal networks to solve an objective. In order to generate different network topologies, neuroannealing stochastically adds and removes new links and nodes to existing networks. In addition, neuroannealing organizes nodes into layers and provides mutation operators to add and remove entire layers of neurons. A layer is a group of nodes such that within a layer, all nodes are of the same type, either inputs, outputs, or hidden nodes. In a layered RNN, links interconnect neural layers, so that two nodes are connected if and only if their respective layers are connected. Links between two layers are associated with a weight matrix containing the connection strengths between the nodes in each layer.

The concept of layers is standard when training neural networks using supervised techniques. Every layered RNN corresponds exactly to a basic RNNs described above. Layers merely add a conceptual separation that is useful for computational efficiency, since it reduces the number of weights that must be stored and multiplied. In neuroannealing, layers also play a role in allowing the structure of the network to expand in useful ways. Neuroannealing probabilistically inserts layers that are designed to store the prior state of another layer, providing a natural way for RNNs to develop an otherwise improbable memory.

To represent a NEAT RNN as a layered network, each node can be assigned to its own layer. The effect on neuroannealing's optimization ability can be tested by enforcing this property on all proposed networks.

13.2.2 Base Measure for RNNs

The measure over RNNs used by neuroannealing is a sum of simpler measures built on top of each other. The space of layered RNNs can be partitioned according to the following four features: (1) the number of layers ℓ, (2) the number of nodes in each layer s, (3) the connectivity pattern among the links c, and (4) the weight values w. A layered RNN representation can be identified exactly by the tuple (ℓ, s, c, w). The base measure will be constructed by addressing each of these items in reverse. The construction of the measure is an important aspect of evolutionary annealing. Since the value of the base measure appears in the selection probability for the next population of networks (Equation 11.9), networks that are preferred by the base measure will be explored more thoroughly. In general, the driving force behind the decisions below is to emphasize smaller, less complex networks without penalizing extra structure too severely.

The first three criteria above comprise the network topology. If ℓ, s, and c are all fixed, then an RNN may be described completely by listing its weights and biases. There are a fixed number of weights and biases, and so an RNN with a given topology may be treated as a vector in $\mathbb{R}^C$ where $C = C(c)$ is the number of weights and biases. At this point, one could place a bound on the magnitude of the weight and use the Lebesgue measure, as was done for REA. Instead, neuroannealing utilizes a Gaussian measure to allow unbounded weights with a preference for small weights. For a given ℓ, s, and c, then, the measure over RNNs matching this profile is given by

$$\lambda_{\ell,s,c}(A) = \int_A \exp\left(-\frac{x^2}{2\gamma^2}\right) dx \tag{13.2}$$

for $A \in \mathcal{B}[\mathbb{R}^C]$. The factor γ is termed the *space scale*; it reflects the average absolute correlation between connected nodes. A good default for the space scale is $\gamma = 1$.

Next, networks with the same number of layers and layer sizes but different connectivity are handled. The connectivity pattern c can be represented as a binary string of size $L = N^2$ where N is the total number of nodes in the network, $N = \sum_i s_i$. L is the number of possible links. Let $n(c) = \sum_i c_i$ be the number of actual links in c. Given ℓ and s, there are exactly 2^L distinct connectivity patterns. Let P be the set of such patterns. A set A of RNN representations with different connectivity patterns may be partitioned into a finite family of sets $\{A_c\}_{c\in P}$, separating out RNNs by connectivity. A measure over such sets is given by

$$\lambda_{\ell,s}(A) = \sum_{c\in P} \frac{1}{n(c)} \left(\begin{array}{c} L \\ n(c) \end{array}\right) \lambda_{\ell,s,c}(A_c). \tag{13.3}$$

Here the factor $1/n(c)$ is applied to prefer networks with lower connectivity, and hence fewer parameters. The factor $\left(\begin{array}{c} L \\ n(c) \end{array}\right)$ is added to emphasize networks that have about half of the possible number of links. The combined effect of the two parameters prefers smaller networks that possess a reasonable number of links.

If only the number of layers is fixed, the number of sizes s is a vector of positive integers greater than one with dimension ℓ. Networks with smaller layer sizes are preferable, but layers of size one should not be emphasized too strongly, or else neuroannealing will not consider larger layer sizes. This balance was accomplished by weighting each size profile inversely to the total number of nodes in the network. There are countably many possible layer sizes, and these can be enumerated. Let S be the set of size profiles, and define

$$\lambda_\ell(A) = \sum_{s \in S} \frac{1}{\sum_i s_i} \lambda_{\ell,s}(A_s), \tag{13.4}$$

where A_s, like A_c in the last paragraph, decomposes A according to size profiles. It is notable that λ_ℓ is not finite, unlike $\lambda_{\ell,s}$ and $\lambda_{\ell,s,w}$. First, there are many size profiles with equivalent sums, and second $\sum 1/k = \infty$ even if there were not. The theory of evolutionary annealing only applies to finite measures. A finite measure over size profiles can be obtained by capping the total size of the network with some large value. In practice, the experiments in this chapter never produced a network larger than 256 nodes, and so this value was used as a maximum network size.

The base measure over RNN representations is achieved by handling arbitrary numbers of layers. This number is an integer greater than one. As with sizes, a set of RNNs may be decomposed according to the number of layers, so that for a given set of RNNs A, the set A_ℓ is the subset of A with ℓ layers. Then a measure over arbitrary layered RNNs is given by

$$\lambda(A) = \sum_{\ell=2}^{\infty} \frac{1}{\ell} \lambda_\ell(A_\ell). \tag{13.5}$$

Putting it all together,

$$\lambda(A) = \sum_{\ell=2}^{\infty} \frac{1}{\ell} \sum_{s \in S} \frac{1}{\sum_i s_i} \sum_{c \in P} \frac{1}{n(c)} \binom{L}{n(c)} \int_{A_{\ell,s,c}} \exp\left(-\frac{x^2}{2\gamma^2}\right) dx. \tag{13.6}$$

Once again, this measure is not finite, but a finite measure can be obtained by bounding the size of the network at some large value. In the experiments that follow, the number of layers was bounded above by 256; more than 20 layers were rarely observed.

The base measure induces a σ-algebra over the space of RNNs whose structure primarily reflects the underlying Euclidean space of the weights. The space of network topologies is discrete and countable, and so a topology for the space of RNNs can be defined as a countable product topology based on the Euclidean topology over the weights.

13.2.3 Redundant Representations

The space of RNNs is treated as a proxy for searching a functional space. Due to Cybenko's density result [49], the space of RNNs spans at least $L^1[\mathbb{R}^d]$. This relationship is not a formal isomorphism, however. There may be many RNNs that compute the same function, even if the number of hidden nodes is fixed. Equivalent RNNs can be generated by swapping the connection strengths between equivalently connected RNNs. Thus even at a basic level, RNN representations are not unique in the sense of computing equivalent functions. The addition of layers introduces further potential for different representations of the same function.

This non-uniqueness does not prevent searching for neural networks, but it is an issue to consider in neuroannealing for two reasons. First, the space of RNN representations contains numerous redundancies. Functions with many representations will be assigned greater mass than functions with fewer representations, meaning that neuroannealing will be more likely to select such functions. Second, the objective value of different representations of equivalent functions is the same, and an information-maximizing optimizer should utilize this information in order to narrow the search space as quickly as possible.

At this time, it is not clear how to structure the base measure to eliminate redundancy and propagate objective evaluations among different network topologies. A non-redundant measure would likely improve performance, especially when searching among complex networks. Such a direction is left as future work.

13.2.4 Partitioning Networks

Evolutionary annealing works by partitioning the search space at increasingly fine resolution one point at a time. There are many ways in which such partitioning could be done. Neuroannealing extends the basic partitioning algorithm in Algorithm 2 (Section 11.2.3) to account for differences in network topology. For this purpose, the partition tree is conceptually stratified into four sections, one for each of the four levels used to define the base measure in Section 13.2.2.

The stratification can be best understood by starting with the node-separation algorithm. Given two networks x_1 and x_2 and a set A, neuroannealing must create disjoint sets A_1 and A_2 such that $x_1 \in A_1$ and $x_2 \in A_2$. The networks can be decomposed so that $x_i = (\ell_i, s_i, c_i, w_i)$ for $i = 1, 2$. If $\ell_1 \neq \ell_2$, then compute the midpoint $\tilde{\ell} = \lceil \frac{\ell_1+\ell_2}{2} \rceil$, and let A_1 be the set of networks in A with less that $\tilde{\ell}$ layers, and let $A_2 = A \setminus A_1$. This process is a straightforward application of the vector separation method in Section 11.2.3. If $\ell_1 = \ell_2$ but $s_1 \neq s_2$, then the vector separation method can be applied to the size vectors s_1 and s_2. The same approach can also be applied if $c_1 \neq c_2$, and finally if $w_1 \neq w_2$. This approach to separation assumes a hierarchy of separation levels, so that ℓ is separated first, then s, then c, and finally w.

Provided that any traversal through the partition tree from the root respects the ordering of this hierarchy, the tree will correspond to a valid partition. If the ordering is violated, for example, by separating on w at a higher node in the tree, by ℓ at a lower level, and then by w at the the leaf, then the regions contained in distinct branches of the tree may overlap, with deleterious results. Thus a traversal through the tree must be stratified. Any separation on ℓ must occur first, then separation on s, and so on.

Algorithm 2 can be modified to support this stratification by allowing separation at nodes other than the leaves. The network partitioning algorithm for neuroannealing locates the first separating boundary for the new network. If this node is a leaf, then the algorithm proceeds as before using the separation algorithm from this section. But if this boundary occurs at an internal node, then a new internal node must be created, and the point being inserted must be separated from every node under the span of the boundary node. In order to make this approach possible, each node in the partition tree must be marked with the representation (ℓ, s, c, w) that was used to create the node and the index of the tuple that was most recently used to separate the node. Note that the portion of this representation that creates the boundary is shared among all points under the space of the boundary node. For example, if the boundary occurs at s, so that $s' \neq s$ where s' is the size profile of the network being inserted, then it holds that every node underneath the boundary shares the size profile s. By separating s' from s using the vector separation algorithm, the inserted network is partitioned away from every node under the internal boundary node.

The hierarchical partitioning algorithm is given in Algorithm 3. To add a new point to an existing tree, the tree is traversed from the root so long as the representations agree up to the marked separation index. The first node containing the inserted point that disagrees on some part of the representation up to the marked separation index is the boundary node, and is chosen for partitioning. This node is separated as described in this section. The new internal node is marked with the separation index at which the inserted point first disagreed. The branch containing the existing nodes is unchanged. The inserted point is assigned to the other branch and is marked with the representation of the inserted point and separation index 4.

Algorithm 3 Algorithm to Generate a Partition Of RNNs

```
$\{x_m\}_{m=1}^M \subseteq X$, the observed networks as $(\ell, s, c, w)$ tuples
$\mathfrak{T} \leftarrow \{X\}$, the partition tree
$k(i) \leftarrow \emptyset$ for all $i = 1, \ldots, M$, node assignment function
$\mu(\{X\}) \leftarrow (0, 0, 0, 0)$, the node marking function
$idx(\{X\}) = 4$, the node separation index function
for $m \leftarrow 1$ to $M$ do
    $N \leftarrow$ highest node in $\mathfrak{T}$ s.t. $x_m \in N$ and $\exists i \leq idx(N)\, s.t.\, \mu(N)_i \neq x_{m,i}$
    if $\exists j \neq m$ s.t. $k(j) = N$ then
        $N_0, N_1 \leftarrow separate\,(x_j, x_m, N)$
        $\mathfrak{T} \leftarrow \mathfrak{T} \cup \{N_0, N_1\}$
        $k(j) \leftarrow N_0$, $k(m) \leftarrow N_1$
        $\mu(N_0) \leftarrow \mu(N)$, $\mu(N_1) \leftarrow x_m$
        $idx(N_0) \leftarrow idx(N)$, $idx(N_1) \leftarrow 4$
        $idx(N) \leftarrow$ the minimum $i$ s.t. $x_{m,i} \neq \mu(N_0)_i$
    else
        $k(m) \leftarrow N$
        $\mu(N) = x_m$
        $idx(N) \leftarrow 4$
    end if
end for
```

The basic partitioning algorithm introduced in Section 11.2.3 and used for Euclidean space in Chapter 12 maintained a partition tree that represents the entire area of the search space. In contrast, the hierarchical partitioning method only represents the area of the network topologies discovered at each point during execution. When neuroannealing is initialized, the area of the first topology inserted into the tree is used to compute the area of the whole tree for sampling purposes. Thus if the first point is $x_1 = (\ell_1, s_1, c_1, w_1)$, the partition tree is assigned the initial area $\lambda_{\ell_1,s_1,c_1}(X_{\ell_1,s_1,c_1})$. Whenever a point with a distinct topology is encountered, say, $x_2 = (\ell_2, s_2, c_2, w_2)$, then the new node for this topology is assigned the area $\lambda_{\ell_2,s_2,c_2}(X_{\ell_2,s_2,c_2})$. Thus the total area of the partition tree is increased whenever a new topology is inserted. This increase is ignored for the purpose of sampling, as though the area of the new topology had always been present, uniformly distributed among the existing leaf nodes. Since sampling from the tree is normalized, this effect is invisible.

Because the area of new topologies is only added to the partition when a new topology appears, the new area only needs to be propagated up the partition tree. Insertion into the score tree is done as for any other point. Thus the approach of adding new area as topologies are discovered avoids an otherwise troublesome problem of reallocating area from existing nodes in the partition and score tree.

As a result, when a new topology appears, it immediately acquires substantial area, forcing some exploration of the new topology. This effect parallels the use of speciation in NEAT, but is a natural mathematical property of the hierarchical partitioning method.

The hierarchical partitioning algorithm can be easily generalized to other search spaces where there is a hierarchy of criteria useful for partitioning points.

13.2.5 Network Mutations

Once neuroannealing has selected a network to mutate, a sequence of mutations is applied to modify the network. Eight types of mutation are employed, in the following order: (1) uniform crossover, (2) addition of a hidden layer, (3) removal of a hidden layer, (4) addition of a node to a hidden layer, (5) removal of a node from a hidden layer, (6) addition of a link between any two unconnected layers, (7) removal of an existing link, and (8) mutation of the weights with an area-sensitive Gaussian.

After selecting a network, neuroannealing applies crossover with probability 0.5. Crossover combines two networks to form a third network that shares properties of the two parents. When crossover is used in neuroannealing, a second network is selected independently of the first using annealed tournament selection. The structure of the networks is aligned according to the indices of their layers, then the weights from any shared links are recombined using either uniform crossover as in Equation 4.17 with probability 0.6 or intermediate crossover as in Equation 4.26 with probability 0.4. The combined network retains the topology of the first parent, but integrates weights and biases from the second parent where they share structure.

In the context of neural networks, crossover is useful because networks are naturally modular. A subset of weights or structure from a network can increase the objective value of the network independent of the other network parameters. Ideally, crossover would be performed by identifying different modules within the network and creating a new network by recombining the modules from successful networks. In the present case, it is not clear how to identify such modules, and so neuroannealing randomly chooses weights from one or the other parent. Preliminary experiments suggest that the use of crossover on about half of the population improves neuroannealing.

After crossover, further mutations are attempted in the order presented below. Only one such mutation is allowed. Once a layer, node, or link has been added or removed, no further structural changes are permitted.

First, neuroannealing adds a *chained layer* to a network with probability 0.01. A chained layer is a layer of hidden nodes that copies an existing layer of the network and adds two links. The first link runs from the copied layer to the chain layer with the identity matrix as the link weight matrix. The second link connects to a random layer in the network other than the chain layer, including possibly the copied layer. If the copied layer was already connected to the target layer, then the weights are also copied from the existing to the new link. Otherwise, the new weights are sampled from a Gaussian with configurable variance $\hat{\sigma}^2$, defaulting to $\hat{\sigma} = 0.01$. A chain layer preserves the prior state of the copied layer into the next step. This mutation was intended to allow the creation of limit cycles within the network, in accordance with the results in [174]. Successive chain layers can quickly add a short-term memory to the RNN that would otherwise be difficult to attain randomly.

Next, if no chain layer was added, neuroannealing deletes a random hidden layer and all of its associated links with probability 0.01. Removing layers allows unneeded structure to be culled once good solutions are located.

If no modifications are made to the network layers, a node is added to a random hidden layer with probability 0.01. The weights and bias for the new node are sampled from a Gaussian using the same variance $\hat{\sigma}^2$ as described above for new layers. The new node's connections are determined by the existing links over the layered structure. If no node is added, a node is removed from a random hidden layer with probability 0.01, and all of its connections are deleted.

The next two structural mutations alter the network connections if no layers or nodes have been mutated. A new link is added between two random layers with probability 0.025. Nothing is done if the randomly selected layers are already connected. Any new weights are sampled from the same Gaussian as is used for adding nodes and layers, with variance $\hat{\sigma}^2$. If no link is added, a random link is removed with probability 0.025. Once again, the link is removed by selecting two random layers. If the layers are not connected, nothing is done. Link removal is performed this way so that it is less likely that links will be removed from a sparsely connected network.

If no structural mutations have been performed, then the existing weights of the network are randomly modified with probability 0.5 using a Gaussian that reflects the structure of the current partition of the space. The partition tree is traversed to obtain the current upper and lower boundaries on the weights of the potentially recombined network. The upper and lower boundaries are used to determine distinct variances for each weight or bias. Let u and ℓ be the upper and lower partition boundaries for the network's weights. Because the weight space is unbounded, these vectors may be infinite on either side. When the upper and lower boundaries are finite, the desired standard deviation for each parameter is half the distance between the upper and lower boundaries. To account for unbounded weights, u and ℓ are modified by using the cumulative distribution of the Gaussian,

$$\Phi_\gamma(z) = \frac{1}{\sqrt{2\pi}\gamma} \int_{-\infty}^{z} \exp\left(-\frac{x^2}{\gamma^2}\right) dz, \tag{13.7}$$

reflecting the warping of the weight space that is also applied by the base measure of Section 13.2.2. The standard deviation for mutating each weight or bias is then given by

$$\sigma_{n,i} = \frac{\Phi_\gamma(u_i) - \Phi_\gamma(\ell_i)}{2\log n}, \tag{13.8}$$

where n is the number of the generation and i is the index of the component within the weight and bias vector as used for partitioning in Section 13.2.4. Each weight or bias is mutated independently. Scaling the variance in this way preserves well-explored parameters, for which the distance between the upper and lower boundaries is small, while forcing exploration of parameters that have not been partitioned much. The extra logarithmic factor is used to compel faster convergence in higher dimensional spaces, as was done in the 25-dimensional experiments on REA in Chapter 12.

13.2.6 Neuroannealing Instantiation

With the previous subsections in mind, the complete neuroannealing algorithm can be stated. Neuroannealing is evolutionary annealing in the space of layered RNNs with annealed tournament selection using using the base measure from Section 13.2.2 and the hierarchical partitioning algorithm of Section 13.2.4. Selected networks are mutated using the chain of mutations described in Section 13.2.5. The hidden and output layers of the RNNs uses hyperbolic tangent activations.

The initial population of networks is sampled as follows. All initial networks have the same topology, which consists of a single input layer and a single output layer, with the input layer fully connected to the output layer. Within this topology, the initial weights and biases are chosen uniformly at random inside $[-\hat{\sigma}, \hat{\sigma}]$ where $\hat{\sigma}$ is the variance to be used when adding layers, nodes, and links. At initialization, the weights are intended to be small so that the activation can quickly change with new mutations, promoting fast exploration of the space.

As presented, neuroannealing has four parameters that must be configured: (1) the population size K, (2) the learning rate η, (3) the space scale γ, and (4) the standard deviation of the components, $\hat{\sigma}$. Based on preliminary experiments, a reasonable set of defaults is $K = 50$, $\eta = 0.1$, $\gamma = 1.0$, and $\hat{\sigma} = 0.1$. The defaults work well for all of the experiments below except for double pole-balancing with reduced inputs, where the values $K = 50$, $\eta = 0.025$, $\gamma = 2.5$ and $\hat{\sigma} = 0.25$ were used instead.

With the algorithm fully described, a set of experiments will be presented to compare neuroannealing with NEAT.

13.3 Neuroannealing Experiments

Experiments were performed in four domains, two in which NEAT performs well, and two in which it does not. The domains and experiments are described below.

13.3.1 Experimental Setup

For the experiments in this section, except as noted otherwise, both neuroannealing and NEAT were run for $1,000$ generations with a population size of 50, totaling $50,000$ evaluations. The parameters for NEAT were set according to the defaults distributed with the publicly available C++ package, except for non-Markov double-pole balancing, where they were set to match [197]. In contrast to previous experiments, each of the tasks below is stated as a maximization problem. To maximize with neuroannealing, the ranking used for tournament selection simply sorts from highest to lowest score rather than the opposite.

Results are reported for each experiment using the performance criteria of Chapter 8: success probability (σ_ϵ^N), hitting time on success ($\hat{\psi}_\epsilon^N$), final error (ζ_{T_n}), average error (ϕ_1), and weighted average error (ϕ_2). The three error-based performance criteria are scaled between 0 and 1 where possible. The error threshold ϵ was chosen separately for each task and is given in the table.

Each task is now described in turn along with its experimental results. The results for all experiments are also compiled and presented in Appendix A as Table A.51.

13.3.2 Double Pole-Balancing

The double pole-balancing task is a control problem in which two poles are attached to a moving cart with hinges. The first pole is 1m in length with mass 0.1kg, and the second is 0.1m with mass 0.01kg. The 10kg cart moves along a track 4.8 meters in length, and must balance the two poles simultaneously by keeping them with 36 degrees of vertical. A motor is attached to the cart that outputs a force up to 10N in either direction along the track at each point in time. Interactions are assumed to be frictionless. This physical system is simulated using a fourth order Runge-Kutta method with state updates every 0.2 seconds. The system starts with the cart in the middle of the track, the smaller pole upright, and the larger pole at 4 degrees from vertical. A successful controller must remain on the track and keep both poles within the tolerance for $100,000$ steps, or about half an hour of real time. The physical equations for the system and further details of the simulation can be found in the literature [79, 80, 197].

The neural network is tasked with controlling the direction of the force and is queried after each state update. Six state variables are available: the position and velocity of the cart, x and $\dot{x}$, the angle and angular velocity of the first pole, θ_1 and $\dot{\theta}_1$, and the angle and angular velocity of the second pole, θ_2 and $\dot{\theta}_2$. There are two versions of this task. In the first version, all six variables are provided to the network, and the network output is scaled to $[-10, 10]$ and applied as the force. This Markov version of the problem can be solved without any hidden nodes. A second, more difficult version of the task provides only the position and angles to the network, requiring the network to infer the velocities over time. This non-Markov version can be solved with as few as two hidden nodes [197].

The objective value of a network for double pole-balancing with or without velocities is the number of steps for which the cart remains on the track with the poles upright. The version with the velocities is termed *Markov*, and the version just the position and angles is termed *non-Markov*.

Table 13.1: Published results for selected methods on both versions of the Double Pole-Balancing task, as given by Gomez et al [80]. Reported quantity is the average number of evaluations before success, with failed trials excluded (i.e., $\hat{\psi}_0^N$, $N = 100,000$ for new results). Results for neuroannealing are new (as indicated by the asterisks), as well as the results for NEAT (determined experimentally using the parameters published by Stanley [197]).

Method	Markov	non-Markov
SANE	12,600	262,700
Q-MLP	10,582	–
Neuroannealing	*7,767	*7,499
ESP	3,800	7,374
NEAT	*1,819	*4,676
CMA-ES	895	3,521
CoSyNE	954	1,249

For NEAT, the parameters for non-Markov double-pole balancing were chosen to match those used by Stanley [197]. Most importantly, the population size for NEAT was increased to 150. For neuroannealing, the settings $K = 50$, $\eta = 0.025$, $\gamma = 2.5$, and $\hat{\sigma} = .25$ were used in place of the defaults to promote larger weights and more thorough exploration of the space.

The Markov version of the task has been solved by a number of methods, including reinforcement learning (Q-MLP), and both versions have been solved by neuroevolution methods (SANE, ESP, NEAT, CMA-ES, CoSyNE) [80]. The non-Markov task is more challenging and has so far only been solved through neuroevolution. The number of network evaluations required to solve the problem is available for each method and can be compared with the results

Table 13.2: Results of neural network experiments on both versions of the double pole-balancing task using the performance criteria of Chapter 8: success probability (σ_ϵ^N), hitting time on success ($\hat{\psi}_\epsilon^N$), final error (ζ_T), average error (ϕ_1), and weighted average error (ϕ_2). The first standard deviation is provided where possible; the value for σ_ϵ^N is accurate within ± 0.005 with $p < 0.05$. The error threshold ϵ was chosen separately for each task and is given in the table. Neuroannealing succeeds on both tasks, but requires more objective evaluations than NEAT.

Neuroannealing

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Double Pole (Markov)	1.0	0.845	7,767 ± 4,871	0.154 ± 0.360	0.203 ± 0.342	0.991 ± 0.071
Double Pole (non-Markov)	1.0	0.960	7,499 ± 3,157	0.039 ± 0.195	0.163 ± 0.181	0.998 ± 0.006

NEAT

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Double Pole (Markov)	1.0	1.000	1,819 ± 2,276	0.000 ± 0.000	0.018 ± 0.044	0.566 ± 0.472
Double Pole (non-Markov)	1.0	1.000	4,676 ± 2,107	0.000 ± 0.000	0.012 ± 0.012	0.742 ± 0.415

for neuroannealing, as is done in Table 13.1. As these results together with Table 13.2 show, neuroannealing is able to solve the pole-balancing task, but requires twice as many evaluations as NEAT does. Neuroannealing takes longer because it searches more thoroughly around previously observed solutions. While such a search is not particularly useful on this problem, it turns out to be valuable on the next two tasks.

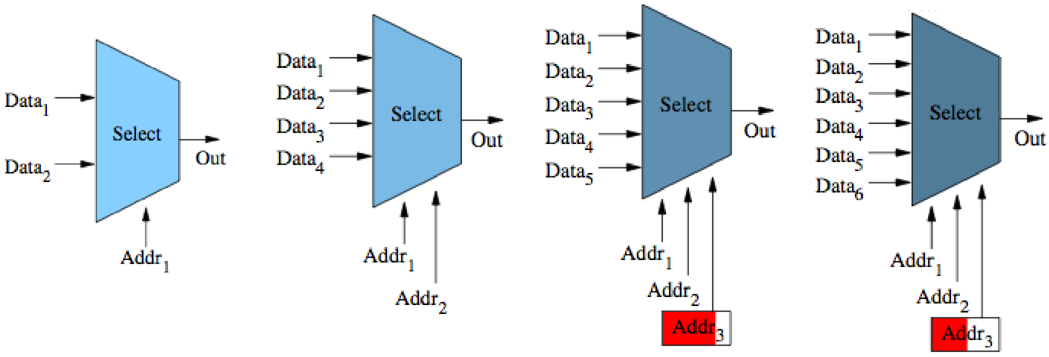
13.3.3 Multiplexers

A multiplexer is a circuit that selects one of several input lines using a binary address. Multiplexers are used to implement computer memory circuits and are easily implemented in hardware. The function of a multiplexer is difficult for a network to learn because it requires the use of a large percentage of the binary input space. A single perceptron can only distinguish a fraction of the binary numbers, and thus multiple neurons must be used in concert to solve the multiplexer problem. As a result, methods like NEAT have difficulty discovering the required complexity [110].

The experiments below test the ability of neuroannealing to learn multiplexers with four different inputs. Mux12 has one address line and four binary inputs. Mux24 uses two address lines and four binary inputs. Mux35 has three address lines and five binary inputs, while Mux36 has three address lines and six inputs. The versions with three address lines use less than the possible eight data inputs in order to simplify the task for neural networks. The task in each case is to learn a network that reads the binary address lines and outputs the binary input at the specified address line. The data inputs are numbered in the standard binary order. Figure 13.1 shows a visualization of the multiplexer problem, taken from Kohl [110].

The objective function (i.e. the fitness function) sums the error at each feasible address and data input. The network outputs are scaled to $[0, 1]$ for this purpose. If $\mathrm{net}(a, d)$ is the scaled output of the network for an address a and a data input d and d_a is the addressed data, the objective is given by

$$f(\mathrm{net}) = \sum_{a,d} |d_a - \mathrm{net}(a, d)|. \tag{13.9}$$



(a) Mux12 (b) Mux24 (c) Mux35 (d) Mux36

Figure 13.1: The multiplexer learning problem. The correct output is determined by the value of the data input at the address specified. With three address bits (Mux35 and Mux36), not all addresses were used to simplify the problem for a neural network.

Importantly, the objective function is structured to maximize the error and hence the learning signal; in practice, it is sufficient to measure the results by checking whether the net output exceeds a threshold.

Table 13.3: Results of neural network experiments on the multiplexer problem using the performance criteria of Chapter 8. The error threshold ϵ for each task is given in the table. Neuroannealing outperforms NEAT on the multiplexer problems in terms of final error; these results are statistically significant ($p < 0.01$).

Neuroannealing

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Multiplexer, 1 × 2	0.010	0.130	15,376 ± 15,930	0.108 ± 0.042	0.113 ± 0.033	0.176 ± 0.050
Multiplexer, 2 × 4	0.200	0.047	19,833 ± 10,351	0.247 ± 0.037	0.252 ± 0.025	0.329 ± 0.023
Multiplexer, 3 × 5	0.250	0.028	20,566 ± 15,509	0.285 ± 0.013	0.287 ± 0.013	0.363 ± 0.017
Multiplexer, 3 × 6	0.300	0.036	17,675 ± 12,449	0.305 ± 0.013	0.308 ± 0.011	0.385 ± 0.012

NEAT

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Multiplexer, 1 × 2	0.010	0.000	50,000 ± 0,000	0.166 ± 0.027	0.180 ± 0.012	0.187 ± 0.000
Multiplexer, 2 × 4	0.200	0.000	50,000 ± 0,000	0.279 ± 0.001	0.282 ± 0.001	0.300 ± 0.008
Multiplexer, 3 × 5	0.250	0.000	50,000 ± 0,000	0.322 ± 0.001	0.325 ± 0.001	0.340 ± 0.007
Multiplexer, 3 × 6	0.300	0.000	50,000 ± 0,000	0.348 ± 0.003	0.351 ± 0.002	0.367 ± 0.005

The results in Table 13.3 show that neuroannealing performs better than NEAT on the multiplexer problems. On 13% of all runs, neuroannealing completely solves Mux12, whereas NEAT was unable to find a solution after 200 runs. The best solution discovered by neuroannealing for Mux24 was also completely correct, although the average solution achieved a fitness of 0.75 against an average of 0.72 for NEAT. On the versions of the problem with three address lines, Mux35 and Mux36, neuroannealing similarly performed well, with an average fitness of 0.72 and 0.70, compared to an average fitness of 0.68 and 0.65 for NEAT. The best fitness in 200 trials for neuroannealing on Mux35 was 0.97, and on Mux36 it was 0.92. The best networks on this task were indeed large. Typical solutions for neuroannealing used 4-6 layers with about 20 nodes. Thus neuroannealing is able to solve the multiplexer problems better than NEAT because it is able to discover complex networks with high objective values that NEAT is unable to reach. The next task, learning concentric spirals, reinforces this point.

13.3.4 Concentric Spirals

In the Concentric Spirals problem [159], the state space is divided into two interlocking spirals, one “black” and the other “white”, and the task is to identify whether each point in the space falls inside of the black or white spiral [110, 159]. The black spiral is determined by 97 points, given in polar coordinates by

$$r_i = \frac{6.5}{104}\left(104 - i\right), \quad \theta_i = \frac{\pi}{16} i \tag{13.10}$$

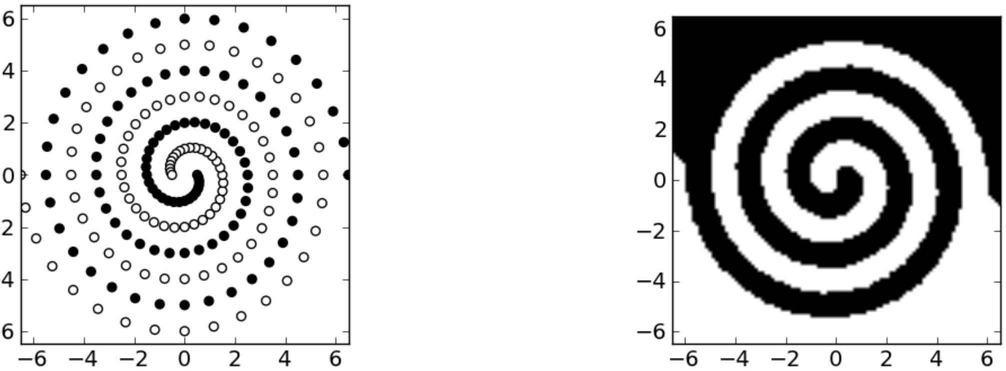
for $i = 0, \ldots, 96$. The white spiral is defined by inverting the sign of r_i. The 194 points are shown in Figure 13.2(a). The spate space is then divided between the two spirals by classifying each point to match the closest spiral. The resulting state space is shown in Figure 13.2(b). An evenly spaced 100×100 grid was overlaid on the state space over the region $[-6.5, 6.5]^2$, and the resulting $10,000$ points were classified in this manner.

The neural network has two inputs and one output. The Cartesian coordinates of the state space are passed to the network as input, and the single output should read 1.0 for black, and 0.0 for white. For this experiment, the objective function summed the errors at each output for every point on the 100 × 100 grid, scaled between 0.0 and 1.0. Thus the sigmoidal outputs of NEAT were used directly, and the hyperbolic tangent outputs of neuroannealing were shifted and scaled as required. If $\mathrm{net}(x, y)$ is the scaled output of the network for the given Cartesian coordinates and $c(i, j)$ is the correct classification for position (i, j) on the grid, then the objective function (i.e. the fitness function) is

$$f(\mathrm{net}) = \sum_{i=1}^{100} \sum_{j=1}^{100} |c(i, j) - \mathrm{net}(x_i, x_j)|. \tag{13.11}$$

It is possible to score a fitness of 0.67 on this problem by learning a correctly angled hyperplane on the state space. To achieve higher scores, the network must learn the spiral structure. Concentric spirals tests the ability of a network to distinguish nearby points in the state space that should be classified differently. In Kohl's terms, the state space is fractured. Such a task requires networks with many nodes to represent the space, which were shown by Kohl to be difficult for NEAT to discover [110].

Experiments were performed for both neuroannealing and NEAT for $1,000$ generations with a population size of 50 and 200 trials. As expected, NEAT performed poorly, rarely exceeding the basic hyperplane solution with fitness 0.67. By contrast, neuroannealing outperformed the hyperplane approximation on about half of the runs, correctly classifying 69% of the points on the average. Complete results are in Table 13.4.



(a) Control Points

(b) State Space

Figure 13.2: Illustration of the Concentric Spirals Problem, in which points must be correctly classified as belonging to interlaced black and white spirals. The left panel shows the 197 control points used to define the problem, and the right panel shows the state space divided according to whether a black or white point is closer. The percentage of correct classifications on the points in the 100 × 100 grid in the right panel was used for training neural networks.

Table 13.4: Results of neural network experiments on the concentric spirals problem using the performance criteria of Chapter 8. The error threshold ϵ is given in the table. Neuroannealing outperforms NEAT substantially in terms of final error; this result is statistically significant ($p < 0.01$).

Neuroannealing

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Concentric Spirals	0.300	0.261	21,687 ± 7,834	0.310 ± 0.021	0.317 ± 0.014	0.333 ± 0.001

NEAT

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Concentric Spirals	0.300	0.000	50,000 ± 0,000	0.331 ± 0.000	0.331 ± 0.000	0.332 ± 0.000

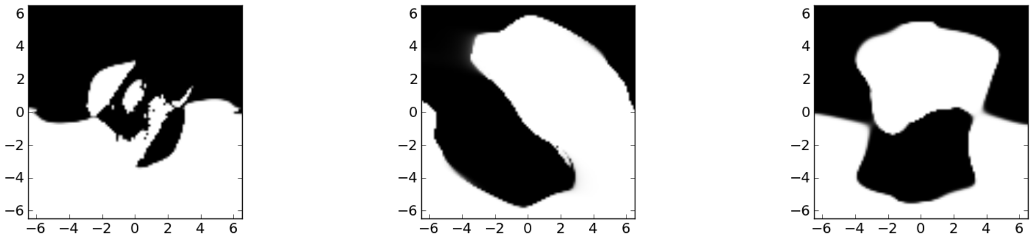
Figure 13.3 shows the learned classifications from several runs of neuroannealing. Over time, neuroannealing eventually discovers solutions that correspond to a spiral shape on the state space. Such solutions generally correspond to larger networks. Only one of the solutions shown in the figure comes from a network with less than 20 nodes. Networks in the figure generally consisted of $4-7$ layers: The largest network, with 77 nodes, had a chained layer of size 37 that allowed correct classification of 30 extra points more than the network without the chained layer. As the networks become larger, they are better able to model the concentric spirals, but the learning progress slows down because larger networks have higher dimension. Neuroannealing was still improving at the end of $50,000$ evaluations ($1,000$ generations), and it is possible that much better networks would have been discovered with more evaluations. In general, it may be conjectured that neuroannealing is more capable of discovering complex solutions in part because annealed selection allows it to follow suboptimal intermediate steps to arrive at more complex optima.

13.3.5 Currency Trading

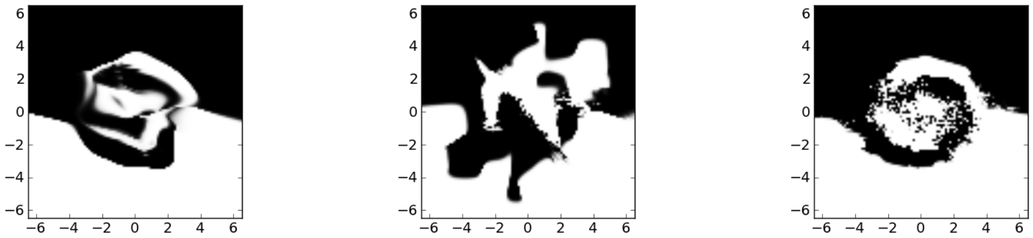
Both neuroannealing and NEAT were also tested on the task of automated currency trading. In this task, a neural network is presented with input data derived from the hourly exchange rate between two currencies. The network must decide which currency to hold each hour and with how much leverage. This task is a new benchmark, introduced in this dissertation. The complexity of the task is unknown, but it is of interest as a real-world problem where optimization of neural networks could prove useful.

The task of an automated currency trader is to progressively read a sequence of technical indicators for an arbitrary currency exchange rate and output trading decisions. Performance on this task was tested using a dataset with six months of hourly exchange rate data for nine different currency pairs. Complete details of the task and the associated datasets are provided in Appendix B.

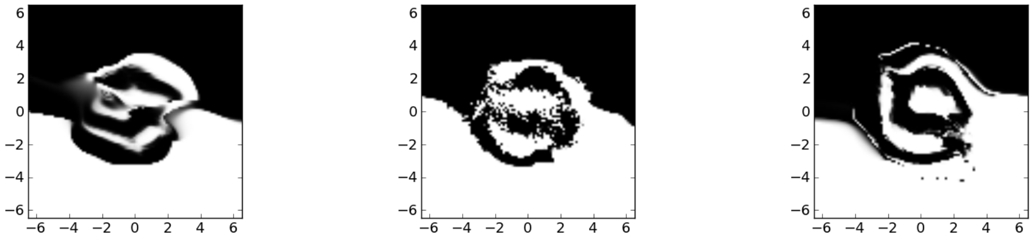
The sequence of technical indicators for this experiment consists of ten real-valued inputs derived from the exponential moving average at five, 20, and



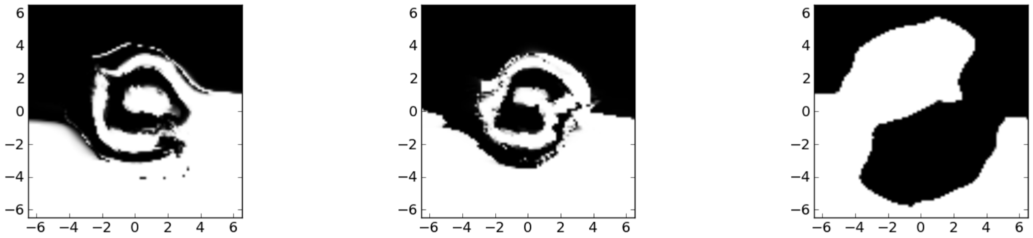
(a) $f = .7222$, 34 nodes (b) $f = .7240$, 21 nodes (c) $f = .7255$, 29 nodes



(d) $f = .7264$, 13 nodes (e) $f = .7277$, 31 nodes (f) $f = .7318$, 39 nodes



(g) $f = .7343$, 20 nodes (h) $f = .7371$, 77 nodes (i) $f = .7385$, 31 nodes



(j) $f = .7389$, 37 nodes (k) $f = .7494$, 30 nodes (l) $f = .7511$, 40 nodes

Figure 13.3: State space classification for the concentric spirals problem as learned by neuroannealing. Objective values and network sizes are shown for each solution. More accurate solutions require larger networks. Neuroannealing is able to discover these solutions, whereas NEAT does not.

50 periods, the relative strength index at 14 periods, the fast and slow stochastics at 14 and three periods respectively, the width of the Bollinger Bands, the position of the closing price within the Bollinger Bands, the absolute difference between the opening and closing price, and the difference between the high and low price. Each of these indicators were scaled to remove the price details, as described in Appendix B.

In addition to these ten technical indicators, three trading inputs were used to describe the automated trader's current position. The first trading input specifies the trader's current long position as a percentage of the possible long value if the trader were operating at full leverage. This input is zero if the trader's position is currently short or neutral. The second trading input gives the trader's current short position as a percentage of the possible short value in the same way. The third trading input provides the length of time the trader's current position has been held under an exponent. If the number of time steps the current position has been open is ℓ, then the value of this input is $\exp(-\ell)$, so that this input exponentially tends toward zero the longer the position is held. These three trading inputs allow the network to be aware of the status of its trades, which is necessary since not every network decision can be implemented, and the simulator may impose a margin call after a bad trade.

The networks for this task have three outputs. The output values are normalized so that the three outputs total to 1. The first output is for buy decisions, and the second for sell decisions. The third output represents risk aversion and is only used to normalize the other two. If the normalized value of first output exceeds the second by 0.05, a buy decision is entered, or a long position is held. If the normalized value of the second output exceeds the first by 0.05, a sell decision is entered, or a short position is held. Otherwise, the current position is liquidated. The amount of leverage is decided as a percentage of the possible leverage (up to 50:1 in currency trading) based on the absolute difference of the normalized buy and sell signals.

The objective value of a currency trader is determined by the account value after trading each of the nine currency pairs in succession for six months. The network starts with an account value of 1 and is queried once per hour on historical data. Objective values less than 1 represent an overall loss, and

Table 13.5: Results of neural network experiments on the currency trading task using the performance criteria of Chapter 8. For currency trading, the global optimal value are unknown, and the values for σ_ϵ^N and $\hat{\psi}_\epsilon^N$ were found by averaging over trials on which the final account value was greater than 250. Neuroannealing and NEAT both perform well, but neuroannealing performs slightly better, achieving higher success probability, higher final account values, and the largest overall account value.

Neuroannealing

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Currency Trading	$f > 250$	0.749	20,054 ± 11,189	31016.331 ± 55094.212	10904.020 ± 20980.836	0.930 ± 0.284

NEAT

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Currency Trading	$f > 250$	0.028	29,425 ± 8,095	43.365 ± 103.898	15.910 ± 48.626	0.033 ± 0.105

values above 1 correspond to a gain. Six months of trading on nine pairs equates to four and a half years of trading time, so a 10% annual gain would result in a fitness of $1.1^{4.5} = 1.54$. A fitness of 10 implies an annual gain of 66%.

Table 13.5 presents the results for this task. Since the true optimum is unknown, the raw fitness values were reported rather than the error. A fitness of 250 or more was considered a success. In the columns for ζ_T, ϕ_1, and ϕ_2, the final fitness, average fitness, and weighted average fitness were reported instead of the standard values for these performance criteria.

On this task, neuroannealing achieved a substantially higher average fitness than NEAT over 200 trials. Neuroannealing attained a fitness of $31,016.331$ on average, versus 43.365 for NEAT. Due to the definition of the fitness, the account value grows exponentially when a neural network trader succeeds, accounting for the large numbers. Neuroannealing also posted higher account values more frequently than NEAT, as reflected in the value for σ_ϵ^N in Table A.51, which shows that neuroannealing attained an account value of 250 or more on 74.9% of all runs, versus 2.8% for NEAT. This difference is statistically significant. The highest fitness value discovered by NEAT was 803 for NEAT, compared with the average fitness for neuroannealing at $31,016$. Both neuroannealing and NEAT performed well on this task overall. It is unclear whether these results will generalize to real-world trading contexts, since it is possible that both algorithms are overfitting the data. Nonetheless, higher objective values suggest greater success in this task. Of the two methods, neuroannealing performs considerably better.

13.4 Neuroannealing Discussion and Future Work

The experiments show that neuroannealing is an effective method for training neural networks in three different domains: multiplexers, concentric spirals, and currency trading. Neuroannealing works well on these problems because it searches more thoroughly through complex networks and is not constrained by population size. Annealed selection makes it possible for neuroannealing to attempt more ways of increasing network complexity without forgetting previously successful solutions. This property allows neuroannealing to step through regions of suboptimal fitness in order to find successful complex networks. When simple solutions exist, neuroannealing usually finds them, because it searches simple networks first. When complexity is required, however, neuroannealing considers progressively more complex solutions.

In double pole-balancing, neuroannealing does not find solutions as quickly as NEAT, ESP, or CoSyNE, but it does solve the problem. This success is achieved despite the fact that neuroannealing is designed to focus on thorough optimization rather than speed. Neuroannealing is a robust optimizer even in domains where NEAT performs well.

On the multiplexer problems and on concentric spirals, neuroannealing performs substantially better than NEAT because it is more capable of discovering complex networks. The size of these networks can exceed those of NEAT networks by a full order of magnitude, as demonstrated by the networks with up to 77 nodes in Figure 13.3. As noted by Kohl [110], these problems require complexity in order to be solved, and neuroannealing is able to deliver.

Neuroannealing thus demonstrates the power of fully leveraging information in order to drive optimization. As an instance of evolutionary annealing, neuroannealing proves that evolutionary annealing can work well in at least some high-dimensional domains and further reinforces the value of the annealed selection methods.

Future research into neuroannealing could focus on determining the effect of the various mutation operators and tuning their parameters. In addition, the good use of chain layers suggest that there may be other large-scale agglomerative combination methods for constructing large neural networks from known modular components. One approach in this direction would be to refine the partition method so that partitions reflect network behavior. Modular networks could then be constructed by merging networks from different behavioral regions. Similar work with NEAT has already yielded valuable results in this direction [125].

For evolutionary annealing in general, the principle that partition regions should reflect meaningful distinctions in the search domain is one that deserves further considerations. The current partitions based on axis-parallel hyperrectangles are a rudimentary tool that could be substantially refined.

Also, it can be argued that methods like DE and CMA-ES are effective because they compress the prior search history into a very brief and compact form. By contrast, evolutionary annealing performs no compression and uses the entire evaluation history. It is likely that there is some useful middle ground. Perhaps the results of evolutionary annealing can be achieved while retaining a summarized version of the evaluation history.

Furthermore, the mutation operators for an evolutionary annealing instance will be more successful if they are better aligned with the function prior from which problems are drawn. In neuroannealing as in NEAT, the progressive addition of network structure is intended to mirror the principle of Minimum Description Length, aligning the search methodology with the general assumptions about the nature of the universal prior, as discussed in Chapter 10. The success of both NEAT and neuroannealing in a wide range of domains bolsters this assumption. Still, it seems that neither neuroannealing nor NEAT fully implements the principles of modularity and locality to a satisfactory degree at this time. More work needs to be performed to quantify and implement these principles.

13.5 Conclusion

Neuroannealing was shown to be an effective optimizer in diverse domains, including pole-balancing, multiplexers, concentric spirals and currency trading. In fractured domains, neuroannealing solidly outperforms NEAT due to its ability to discover larger networks with higher objective values. These results demonstrate that neuroannealing is an effective method for optimizing neural networks.

This chapter brings to a close the experimental portion of this dissertation. Evolutionary annealing has been defined as an information-maximizing approach. It has been tested in Euclidean space and neural network with positive results. It is expected that similar information-maximizing optimizers will continue to prove their usefulness in the near future.

Chapter 14

Disscussion and Future Work

The three main contributions of this dissertation are (1) the formalization of iterative stochastic optimizers for static objective functions, (2) a rigorous analysis of optimizer performance suggesting that general-purpose optimizers exist for real-world problems, and (3) evolutionary annealing as an effective information-maximizing optimizer. In this chapter, the broader implications of each of these results are discussed, and potential avenues for future research are presented.

14.1 Formalization of Optimizers

Although the study of optimization has a long history, this dissertation has examined the space of iterative stochastic optimization methods formally, based on the sequence of evaluation points proposed by each method. This perspective enabled new insights, but the results of this dissertation have only scratched the surface of what is possible within the formal framework adopted here. The next several sections propose different ways in which the formalism could be extended or applied to create new optimizers.

14.1.1 Dynamic and Stochastic Objectives

This dissertation has focused on static objectives only. That is, the objective is given beforehand and does not change during the optimization process. Such an objective was described as an element in $\mathbb{R}^X$. But what if the objective changes over the course of optimization, as is the case with dynamic functions? Or what if the objective is a random function, whose values can be sampled but not measured exactly?

The formal context of this dissertation can be extended to dynamic functions with little change. The space of dynamic functions can be represented as either $\mathbb{R}^{X\times\mathbb{N}}$ or $\mathbb{R}^{X\times[0,\infty)}$, depending on whether discrete or continuous time is needed. An optimizer on this space can be regarded as a function from $\mathcal{T}[X] \times \mathbb{R}^{X\times\mathbb{N}}$ to $\mathcal{P}[X]$. This set is once again a closed, convex subset of a normed vector space with a slightly different norm that takes the supremum over dynamic functions. It seems reasonable to speculate that all of the continuity results from Chapter 5 still hold for continuous functions.

The major changes regard the analysis of performance. A dynamic objective may be viewed as an adaptive environment. Such environments have been extensively studied in the context of evolutionary computation. Competitive coevolution in particular is an example of dynamic objective, in which the objective value of a solution depends on the other solutions being evaluated concurrently. The information-maximizing perspective from Chapter 10 no longer applies as strongly in an adaptive environment, since previous objective evaluations may be stale or irrelevant. With a dynamic objective, forgetting can become strategic if the changes to the environment are slow and unpredictable. Or, it may be advisable for the optimizer to model the dynamics of the environment explicitly if changes are predictable, in which case information maximization may still be useful.

If the objective is stochastic and not just dynamic, the same kind of formalization can be applied with optimizers drawn from $\mathcal{T}[X] \times \mathcal{P}[\mathbb{R}^X] \to \mathcal{P}[X]$. This space is also likely to be a normed vector space, and continuity results probably still apply. But now information must be handled even more carefully, and multiple evaluations of nearby points are required for an optimizer to determine correctly where it should search for better optima.

In either case, a formalism similar to the one proposed here could be applied in order to study the performance of optimizers on dynamic or stochastic objectives analytically. Such formal approaches are likely to yield interesting and practical results.

14.1.2 Alternative Characterizations

From the beginning of Chapter 3, a particular norm was chosen and remained fixed throughout the text. Alternative characterizations of the space of optimizers are possible and may be useful for obtaining further results.

As an example, suppose that the space of optimizers is restricted to those optimizers that are absolutely continuous with respect to some finite positive measure μ. Such a space possesses more internal structure, which allows stronger theoretical results to be achieved. As an example, this assumption gives rise to an inner product space in which optimizers may be projected onto each other geometrically.

Definition 14.1.1. *A generalized optimizer* $\mathcal{G} \in \mathcal{MF}_0\left[X, \mu\right]$ *is absolutely continuous with respect to* μ*, written* $\mathcal{G} << \mu$*, if for all* $t \in \mathcal{T}$ *and* $f \in L^1\left[X, \mu\right]$ $\mathcal{G}[t, f]$ *is absolutely continuous with respect to* μ*, that is, if* $\mathcal{G}[t, f](A) = 0$ *whenever* $\mu(A) = 0$ *for all* $A \in \mathcal{B}_\tau$*.*

If $\mathcal{G} << \mu$, there exists a function $G : \mathcal{T} \times L^1 \to (X \to \mathbb{R})$ such that

$$\mathcal{G}[t, f](A) = \int_A G[t, f](x)\, \mu(dx)$$

as a consequence of the Radon-Nikodym Theorem. The output of the function $G[t, f]$ on a particular trajectory and objective is the *Radon-Nikodym derivative* of $\mathcal{G}[t, f]$ with respect to $\mathcal{G}$. In this section, the notation $D_\mu\mathcal{G}$ will represent the pointwise Radon-Nikodym derivative of $\mathcal{G}$ with respect to μ, so that $G = D_\mu\mathcal{G}$.

An inner product can be constructed from this derivative. Consider the set $\mathbb{S}\bot\mathbb{R} = \bigcup_n \mathbb{R}^{n-1} \times \{t \in \mathcal{T} \mid T_m(t) = |t|\}$. Using the notation from that proof, for $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{MF}_{\text{tr}}$ both absolutely continuous with respect to μ, define

$$\begin{aligned} \left(\mathcal{G}_1, \mathcal{G}_2\right)_m \;=\; & \int_{\mathbb{S}\bot\mathbb{R}} D_\mu\mathcal{G}_1[t_1^{m-1}, y_1^{m-1}](t^m) \quad D_\mu\mathcal{G}_2[t_1^{m-1}, y_1^{m-1}](t^m)\ \mu(dt^m) \\ & \times \prod_{i=1}^{m-1} \frac{\mu(dt^i)}{\mu(X)}\ \mathbb{P}_F\left(F(t^i) = dy^i \mid F(t_1^{i-1}) = y_1^{i-1}\right) dy^i, \quad (14.1) \end{aligned}$$

remembering that $\mu(X) < \infty$ by assumption. The operation $(\mathcal{G}_1, \mathcal{G}_2)$ compares the μ-density of $\mathcal{G}_1$ with the μ-density of $\mathcal{G}_2$ at the m^{th} position in a trajectory randomly selected according to μ with evaluations randomly selected according to $\mathbb{P}_F$. This operation is an inner product:

Proposition 14.1.1. *For $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{MF}_{\text{tr}}$ with $\mathcal{G}_1, \mathcal{G}_2 << \mu$, the operation $(\mathcal{G}_1, \mathcal{G}_2)_m$ is an inner product for all $m \geq 1$. Additionally, $|(\mathcal{G}_1, \mathcal{G}_2)_m| < \infty$.*

Proof. To be an inner product, $(\mathcal{G}_1, \mathcal{G}_2)_m$ must be symmetric and linear in the first argument. Symmetry is obvious from Equation 14.1. Linearity follows from the linearity of the Radon-Nikodym derivative and the linearity of the integral. To obtain finiteness, notice that

$$\nu(dt_1^{m-1}, dy) = \prod_{i=1}^{m-1} \frac{\mu(dt^i)}{\mu(X)} \, \mathbb{P}_F \left(F(t^i) = dy^i \mid F(t_1^{i-1}) = y_1^{i-1} \right) dy^i$$

is a probability measure and

$$(\mathcal{G}_1, \mathcal{G}_2)_m = \mathbb{E}_\nu \left[\int_X D_\mu \mathcal{G}_1[t_1^{m-1}, y_1^{m-1}](t^m) \; \mathcal{G}_2[t_1^{m-1}, y_1^{m-1}](dt^m) \right].$$

Finiteness will follow from fact that the term inside the expectation is finite on all trajectories and objectives. $\mathcal{G}_2 << \mu$ and $D_\mu \mathcal{G}_1$ is μ-integrable, so $D_\mu \mathcal{G}_1$ is $\mathcal{G}_2$-integrable. That is, the inner term is finite, so $|(\mathcal{G}_1, \mathcal{G}_2)_m| < \infty$. $\square$

The complexity of Equation 14.1 can be hidden by defining a measure

$$\kappa_m(dt, dy) = \mu(dt^m) \; \nu(dt_1^{m-1}, dy)$$

using the measure ν from the proof of Proposition 14.1.1. If μ is a probability measure (and $\tilde{\mu} = \mu / \mu(X)$ always is), $(\mathcal{G}_1, \mathcal{G}_2)_m$ is just the expectation of the products of the Radon-Nikodym derivatives with respect to κ_m,

$$(\mathcal{G}_1, \mathcal{G}_2)_m = \mathbb{E}_{\kappa_m} \left[D_\mu \mathcal{G}_1 \, D_\mu \mathcal{G}_2 \right].$$

As an inner product, $(\mathcal{G}_1, \mathcal{G}_2)_m$ only compares the m^{th} unique point of a trajectory. To account for all points in a trajectory, fix M as a large finite number and define

$$(\mathcal{G}_1, \mathcal{G}_2) = \frac{1}{M} \sum_{m=1}^{M} (\mathcal{G}_1, \mathcal{G}_2)_m.$$

The following proposition is a trivial consequence of Proposition 14.1.1.

Proposition 14.1.2. $(\mathcal{G}_1, \mathcal{G}_2)$ *is an inner product on* $\mathcal{MF}^{\mu}_{\mathrm{tr}}$*, and* $|(\mathcal{G}_1, \mathcal{G}_2)| < \infty$*.*

Let $\kappa = \frac{1}{M}\sum_{m=1}^{M}\kappa_m$, and as before, if μ is a probability measure,

$$(\mathcal{G}_1, \mathcal{G}_2) = \mathbb{E}_{\kappa}\left[\, D_{\mu}\mathcal{G}_1\, D_{\mu}\mathcal{G}_2 \,\right].$$

The space of optimizers in $\mathcal{MF}_{\mathrm{tr}}$ that are absolutely continuous with respect to μ is therefore an inner product space.

This inner product space is an alternative way of analyzing the space of optimizers, one that places a geometry over optimizers and should permit stronger theoretical results. In some sense, though, the norm used throughout this dissertation and the inner product above are somewhat unnatural, since the values of both of them depend on what an optimizer does with evaluation points it may never see. An ideal characterization would be related to the performance of the optimizer, since it would organize optimizers according to their practical utility for particular problems. The seed of such a characterization is visible in the discussion of performance-based linear projections of optimizers in Chapter 10. There remains substantial work that can be done to elucidate the meaning and practical utility of these concepts.

One aspect of the formalism that was defined but not emphasized was the role of computability. Since only computable optimizers can be run on a digital computer, computable optimizers are an important class of problems. Among other properties, computable optimizers must be information restricted, or else they would not halt. Additionally, programs that halt must have finite length. There are only countably many such programs. Thus the space of computable optimizers is countably infinite. One can choose stochastically between programs, but the space of stochastic computable optimizers with uncountable precision still has a countable basis. The space of computable optimizers is therefore much smaller than the space of all information-restricted optimizers, a fact which may lead to new insights upon further study.

14.1.3 Convex Control of Optimizer Portfolios

Chapter 3 proved that convex combinations of computable optimizers are also computable optimizers. Chapter 7 demonstrated that performance varies nonlinearly as the convex combination changes, and some of the results in Chapter 8 even suggested that convex combinations may outperform any of the optimizers being combined.

In light of these facts, can one choose a good optimizing strategy by convexly combining existing optimizers? One way to do so is to test several strategies and allocate resources to the strategies that perform best. This approach is termed *convex control* of optimizers; it is explored theoretically in this section.

Suppose that there is a finite set of trajectory-restricted optimizers $\mathfrak{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_m\} \subseteq \mathcal{O}_{\text{tr}}$, each of which is known to perform well on a versatile set of function priors. Such a set will be termed an *optimizer portfolio*, and it might include general-purpose methods such as simulated annealing, hill climbing with random restarts, or differential evolution. The convex control problem can be stated as follows: Given a set of optimizers, a prior $\mathbb{P}_F$, and a performance criterion ϕ, choose a time-varying probability vector $\alpha(n) = \alpha_1(n), \ldots, \alpha_m(n)$ with $\sum_i \alpha_i(n) = 1$ for all n such that $\mathcal{G}_\alpha[t, f] = \sum_i \alpha_i(|t|)\mathcal{G}_i[t, f]$ minimizes $g(\tilde{\alpha}) = \langle \mathcal{G}_{\tilde{\alpha}}, \mathbb{P}_F \rangle_\phi$.

At each time step, $\mathcal{G}_\alpha$ is a convex combination over the optimizer set $\mathfrak{G}$. As a result, $\mathcal{G}_\alpha$ can only depend on evaluations along the trajectory. So $\mathcal{G}_\alpha$ is trajectory-restricted. As a function of time, $\mathcal{G}_\alpha$ moves along a trajectory contained in the convex span of $\mathfrak{G}$. Because the probability vector changes with time, $\mathcal{G}_\alpha$ itself cannot be expressed directly as a convex combination over $\mathfrak{G}$.

The convex control problem can be addressed from two perspectives. In the first, the goal is to find a single, stable convex combination that is adapted to a given objective function. In the second, a dynamic control procedure is sought that makes the most efficient use of the optimizer set. Both of these directions are interesting lines of research for future work.

For now, suppose that the probability vector α does not vary with time, i.e. $\alpha = \alpha_1, \ldots, \alpha_m$ independent of the length of the trajectory. Then $\mathcal{G}_\alpha$ is a single convex combination over $\mathfrak{G}$ contained within the convex span of $\mathcal{G}$. A convex combination of optimizers can be regarded as a choice over optimizers. At each time step, the probability vector α is sampled to choose one of the

$\mathcal{G}_k$, and then $\mathcal{G}_k$ is sampled to choose the next evaluation point. Section 3.3.3 introduced the terminology a *history* of $\mathcal{G}_\alpha$ as the sequence of such choices, e.g. $\mathcal{G}_3\mathcal{G}_1\mathcal{G}_4\mathcal{G}_1\mathcal{G}_2\mathcal{G}_2\mathcal{G}_2\mathcal{G}_3\dots$. The set of all histories of $\mathcal{G}_\alpha$ may be regarded as the set of optimization strategies available to $\mathcal{G}_\alpha$ within the Optimization Game of Chapter 10.

Since the objective strategy $\mathbb{P}_F$ is fixed, the second player may be regarded as a chance node and the set of histories of $\mathcal{G}_\alpha$ can be thought of as tracing out an m-ary game tree with chance nodes omitted. A fundamental question is whether convex combinations over a set $\mathfrak{G}$ can outperform the best element in $\mathfrak{G}$. If $\mathfrak{G}$ contains just two members, e.g. $\mathcal{G}_1$ and $\mathcal{G}_2$, then the game tree is a binary tree. The question then resolves to whether one of the two outer histories, $\mathcal{G}_1\mathcal{G}_1\mathcal{G}_1\mathcal{G}_1\dots$ or $\mathcal{G}_2\mathcal{G}_2\mathcal{G}_2\mathcal{G}_2\dots$, outperforms all other histories.

Each choice in this binary tree can be represented as a zero if the left branch is followed, and a one if the right branch is followed. A single history contains infinitely many such choices. Thus the set of histories corresponds to a binary representation of the real numbers between zero and one and is therefore uncountable. On this basis, it would be surprising if the outer two histories were the only two interesting ones from the perspective of performance.

Recall from Section 10.2 that the performance criterion $\phi(\mathcal{G}, F) = \langle \mathcal{G}, \mathbb{P}_F \rangle_\phi$ is linear over $\mathfrak{A}\left[\mathcal{O}_{\mathrm{tr}}\right]$. However, this linearity only applies to convex combinations that are sampled once for the entire history. Thus a convex combination in $\mathfrak{A}\left[\mathcal{O}_{\mathrm{tr}}\right]$ is a single choice between the two outer histories mention above. But $\mathcal{G}_\alpha$ is a convex combination over $\mathcal{O}_{\mathrm{tr}}$, since it makes one choice at each time step. As discussed in Section 7.2.1, $\phi(\mathcal{G}, F)$ is non-linear over $\mathcal{O}_{\mathrm{tr}}$. If $\phi(\mathcal{G}, F)$ were linear over $\mathcal{O}_{\mathrm{tr}}$, then only the outer histories could be optimal. Because it is non-linear, it is possible that one of the uncountably many internal histories could perform best.

Research on applying algorithm portfolios to optimization has been performed by Silverthorn and Miikkulainen [187] with promising results. The discussion above provides further theoretical basis for this research and places it within the context of general-purpose optimization.

14.1.4 Formalization Conclusion

As this section has demonstrated, the formal perspective in this dissertation opens up a new way of looking at optimization methods and provides numerous starting points for future research. This section suggested further research into non-static objectives, alternative formal representations, and convex control. Several other directions are possible as well. For example, how do common analytic notions such as compactness, integrability, and differentiability apply to the space of iterative optimizers? As another idea, it might be possible to conceive of optimizers that operate in continuous time. Such optimizers might be approximated by contracting or dilating the time scale during optimization depending upon the volatility of the optimization trajectory. The breadth of each of these topics taken independently suggests that the formal perspective of this dissertation is a useful tool for studying optimization.

14.2 General-Purpose Optimizers

The NFL Identification Theorems were presented in Chapters 9, proving simultaneously that NFL still applies in arbitrary measure spaces, but that an NFL makes learning impossible by design. Given that learning does occur in the real world, it is reasonable to conjecture that the set of general problems encountered in reality are not subject to NFL. As stated, these results did not apply to gradient-based methods; it would be interesting to know whether the results also apply to broader domains.

14.2.1 Extending NFL to Information-Restricted Optimizers

The NFL results in this dissertation pertained to trajectory-restricted optimizers. Do the same results apply to information-restricted optimizers as well? In fact they do, provided that the information function is fixed. This section sketches the mechanisms for applying NFL type results to $\mathcal{O}_{\text{ir}}$. The proofs in this section will describe how to generalize the results from previous sections while leaving many details for future work.

The main problem in dealing with $\mathcal{O}_{\text{ir}}$ is that the information functions are not shared among optimizers. The definition of $\mathcal{O}_{\text{ir}}$ was based on the existence of an information function $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$. Each optimizer may determine for itself the information it wishes to obtain from function evaluation. An extension of the NFL Identification Theorem and the duality of priors and optimizers can be obtained for $\mathcal{O}_{\text{ir}}$, but the results must be qualified to account for distinct information functions. Recall from Section 10.2 the set $\mathcal{O}_{\text{ir}}^I$ consisting of all information-restricted optimizers compatible with the information function I.

Definition 14.2.1. *A random variable F over $\mathbb{R}^X$ (or its function prior $\mathbb{P}_F$) is information-path independent of an information function $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$ if for any $x \in X$, $F(x)$ and $I(F, x)$ are separately and jointly independent of $F(y_1), \dots, F(y_n)$ and $I(F, y_1), \dots, I(F, y_n)$ for any sequence $y_1, \dots, y_n \in X$ such that $x \neq y_i$ for all i.*

Information-path independent priors do exist. In particular, any NFL prior is information-path independent on the information function $\tilde{I}(f, x) = f(x)$ by the NFL Identification Theorem. Because of the explicit reference to the information function, the concept of information-path independence only allows the NFL Identification Theorem to be expanded to subsets of $\mathcal{O}_{\text{ir}}$ that share a specific information function.

Theorem 14.2.1. NFL Identification Theorem (Extended). *Given an information function $I : \mathbb{R}^X \times X \to \mathcal{T}[\mathbb{R}]$, a function prior $\mathbb{P}_F$ over $\mathbb{R}^X]$ is strongly NFL on $\mathcal{O}_{\text{ir}}^I$ and ζ_{T_m} if and only if $\mathbb{P}_F$ is information-path independent on I and identically distributed on any finite collection of points.*

The proof of the Extended NFL Identification Theorem is broadly analogous to the proof of the NFL Identification Theorem for $\mathcal{O}_{\text{tr}}$. The main difference is that the information function must be included in the probability of the function prior, replacing the function evaluation itself, i.e.

$$\prod_{i=1}^{|t|} \mathbb{P}_F\left(I\left(F, t^i\right) \mid I\left(F, t^j\right) \forall j < i\right) = \prod_{i=1}^{|t|} \mathbb{P}_F\left(I\left(F, t^i\right)\right). \tag{14.2}$$

The remaining logic of the proofs is nearly identical given this change.

It would be tempting to conclude from the extended NFL Identification Theorem that a similar theorem holds over all of $\mathcal{O}_{\text{ir}}$. Such a conclusion is illegitimate, however, since a particular prior may be information-path independent on one information function but not on another. Thus there is no obvious definition that generalizes information-path independence across all of $\mathcal{O}_{\text{ir}}$.

This fact suggests that it may be possible to avoid NFL by switching information functions (and therefore necessarily altering the optimizer). At this time, such a claim is merely speculative, but it seems intuitively plausible. Importantly, an NFL prior does not necessarily exist for an arbitrary information function. However, if the information function is of bounded information (as defined in Section 10.2, then an NFL prior does exist. Hence gradient-based methods are subject to NFL. Unbounded growth of information is required in order to prevent an NFL prior from being constructed.

The weak version of the NFL Identification Theorem also applies to $\mathcal{O}_{\text{ir}}^{I}$ through the use of uncorrelated information-paths defined analogously to information-path independence.

It is an interesting question to ask whether an NFL result is possible for arbitrary subspaces of $\mathcal{PF}$. In general, one might think of each optimizer as having a *knowledge function* that encapsulates the information it is able to discover about an objective given an evaluation history. This knowledge function would differ from an information function in that its cardinality could be arbitrary, where information functions can only return finitely many real numbers. In computational terms, the knowledge function is just the computational state of the optimizer. Every optimizer in $\mathcal{PF}$ should possess a knowledge function that completely describes it. For example, the omniscient optimizer has as its knowledge function a function mapping every objective to its optima. A strongly NFL prior for an arbitrary class of optimizers would need to be constructed explicitly to confound the knowledge function. Whenever it is possible to do so, an NFL result could be obtained.

14.2.2 General-Purpose Optimization Conclusion

The NFL Identification Theorem formalizes certain objections to NFL that have been made over the years, and shows that for large search domains, NFL violates the principle of Occam's razor. This result opens up several avenues of research, particularly regarding the nature of an optimal optimizer for a given function prior. In addition, the nature of the true prior governing reality is unknown in general, and thus a good general-purpose learner ought to be designed to handle this uncertainty about the true prior against which it is optimizing. Several theoretical approaches to solving this problem are possible, and the perspectives developed in this dissertation are expected to help identify them.

14.3 Martingale Optimization

Chapter 10 proposed the information-maximization principle, that the optimal optimizer should fully utilize all available information about the objective function. This principle was used to derive evolutionary annealing as a martingale optimization method. There remains substantial future work to verify the theory of information maximization. Additionally, several improvements to the evolutionary annealing method may be possible as well.

14.3.1 Proving the Optimality of Information Maximization

In Section 10.4.4, it was conjectured that a particular information-maximizing strategy may be optimal, given by

$$Z_{n+1}^{\mathrm{opt}} = \mathrm{argmin}_{x \in X}\ \mathbb{E}_{\mathbb{P}_F}\left[h(Z) \mid \mathcal{H}_n, Z_{n+1}^{\infty} = x\right] \tag{14.3}$$

for a function prior $\mathbb{P}_F$, a performance criterion $\phi(\mathcal{G}, f) = \mathbb{E}_{\mathcal{G}f}\left[h(Z)\right]$, and an evaluation history $\mathcal{H}_n$.

Further consideration may lead to a proof that this strategy or a similar one is theoretically optimal. In general, optimality is necessarily tied to a particular performance criterion. In order to prove that such a strategy is optimal, one might utilize the fact that the conditional expectation is the estimate of a particular random quantity that minimizes the variance. Additionally, it might be useful as a subgoal to prove that the conditional expectation of the performance criterion is a submartingale and that Z^{opt} is the minimum such martingale. A proof of the optimal optimization method would be a valuable result legitimating the line of research adopted in this dissertation.

14.3.2 Semantic Partitioning

In evolutionary annealing, the partitioning method was used to determine how annealed selection apportions probability mass among previously observed evaluation points. For Euclidean space, REA employed axis-parallel hyperrectangles as an efficiently computable partitioning approach. Neuroannealing generalized this partition method to use hierarchical partitioning on a larger space, but at its base, neural networks were also partitioned using axis-parallel hyperrectangles to separate the weights.

What if the partitions of the search space could be arranged to match the natural structure of the objective function? Such semantic partitions might be better able to locate the optima of an objective function by allocating probability more efficiently among the different regions. In a way, hierarchical partitioning as used by neuroannealing is a rudimentary step in this direction. Additionally, it might be possible to recognize fractal structure within semantic regions, and to propagate this structure across different partition regions in order to build a more accurate model of the objective function. For example, the RNN space in neuroannealing contains redundant network representations. If the similarities between two network topologies could be identified, then the objective evaluations from one network topology could be used to estimate the fitness structure of networks in the other network topology without additional objective evaluations. This type of approach could substantially improve the accuracy of evolutionary annealing.

14.3.3 Applications to Other Search Domains

It was mentioned in Chapter 11 that evolutionary annealing has been tested in several domains, including bit strings, structure-learning for Bayesian networks, and game-playing strategies. In each of these domains, competitive results were obtained on benchmark problems. Nonetheless, there is substantial effort involved in applying evolutionary annealing to a new domain. A partitioning method must be developed, along with a base measure and a set of effective mutation distributions. Further experiments in other domains will promote the development of a generalized methodology for instantiating these objects.

14.3.4 Information Compression

Evolutionary annealing retains the complete results for every objective evaluation it performs. The requirements to store this data are manageable, but they also introduce substantial overhead. In addition, computing the next point with evolutionary annealing requires logarithmic rather than constant time in terms of the number of previously evaluated points. It would be desirable to reduce or eliminate this overhead where possible.

As a martingale method, evolutionary annealing is primarily concerned with preserving the full information provided by prior evaluations. However, it is possible that the complete information or a nearly complete approximation can be achieve by compressing the previously evaluated points into a smaller representation. A compressed representation would also have the benefit of generalizing the information learned from the previous evaluations so that objective evaluations are chosen more efficiently in regions where they are more likely to improve performance.

14.3.5 Information Maximization in Stochastic and Dynamic Environments

The information-maximization principle was formulated in the context of static fitness functions. As discussed in Section 14.1.1, if the environment is dynamic or stochastic, then the information-maximizing approach requires some alterations. In the case of stochastic domains, selection of partition regions could be performed in a way that selects a larger region higher in the tree containing several points, effectively averaging over the points in order to avoid committing to a point with spuriously optimal fitness.

In adaptive environments, the compression of information might become especially important if the dynamics of the environment are predictable. In this case, it might work well to extract a set of invariant principles governing the dynamics and to use these principles in conjunction with the observed objective values to determine which points to explore next. If the environment is unpredictable, then some form of strategic forgetting may be helpful. Evolutionary annealing with strategic forgetting would become similar to evolutionary algorithms in which individuals have a “lifetime” that might span several generations.

14.3.6 Information Maximization Conclusion

Overall, the information-maximization principle presents a fresh perspective on optimization that yields several interesting paths for future work. Methods such as semantic partitioning and information compression could result in powerful new optimization techniques. Mutation operators for evolutionary annealing that capture the regularities of general-purpose function priors may also produce substantial advances in optimization technology that are more capable of searching high-dimensional to find elegant solutions for important problems.

14.4 Conclusion

Overall, the most significant aspect of this dissertation is to suggest that general-purpose optimization is not only possible, but can be performed in an optimal way. The concepts of information maximization and martingale methods are introduced as examples of how general-purpose learning might be approached. But the important point is that an effective optimizer may be derived from first principles by considering what sort of function prior governs the sorts of problems that occur in the real world. A robust optimizer can then be developed by considering the path dependencies that arise when a particular function prior is assumed.

In fact, this is exactly the process by which human research is carried out. A problem is analyzed to observe its inherent nature and internal regularities. Then, a hypothesis is generated that encapsulates a proposed compression of the problem into a simpler representation. Such a hypothesis may then be tested and either rejected or further refined. The information obtained through the testing process is added back to the body of observations subject to analysis, and the process continues until a suitable solution is found.

There is no reason why a general-purpose optimizer cannot employ these same principles to solve complex problems. The key point is that the proposed hypotheses must correctly match the universal prior that governs reality. Such a prior should be characterized by local regularity, sparsity, modularity, repeated and analogous structure, and other principles that are observed in the physical world. Information-maximizing optimization strategies that fully utilize their prior observations together with these principles should be capable of finding substantially better solutions than is possible at the current time.

Chapter 15

Conclusion

This dissertation has studied optimization methods from a formal perspective. This perspective made it possible to uncover several important insights into how existing optimization methods may be compared to each other theoretically and experimentally and how the optimization task may be performed optimally. This final chapter reviews the significant contributions of the dissertation and offers concluding remarks to summarize the perspective on general-purpose optimization that has emerged as a result of this work.

15.1 Contributions

As discussed in Chapter 14, this dissertation makes three main contributions to the study of optimization: (1) the development of a mathematically formal approach to iterative stochastic optimization, (2) the discovery of the NFL Identification Theorems and the recognition that general-purpose learning is possible, and (3) the introduction of information-maximizing optimizers such as evolutionary annealing. In this section, each of these contributions is discussed in turn.

15.1.1 Significance of the Formal Approach

While optimization has been studied for centuries, the analytic relationships between distinct optimization methodologies has previously received little attention. In this dissertation, arbitrary optimizers for static functions were studied from a functional analytic point of view, with some surprising results.

In Chapter 3, iterative stochastic optimizers were formalized based on the evaluation points they propose given a particular objective and an evaluation history. This formalization revealed that the space of iterative stochastic optimizers is a closed, convex subset of a normed vector space. Interestingly, this result still holds true when optimizers are considered based upon the infinite sequence of evaluation points they propose for a given objective, as shown in Chapter 6.

The profusion of optimization methods and the clear distinctions in the way they have been presented would lead an observer to initially conclude that methods such as gradient descent and Monte Carlo optimization have nothing to do with each other. The results of this dissertation instead imply that between any two optimizers there is a line in optimizer space that smoothly transforms one optimizer into the other. This fact was explored experimentally in Chapter 8, where it was shown that in many cases the performance of the optimizers along that line changes continuously as well.

The proposed methodology for formalizing optimizers is fully general. Chapter 4 demonstrated that the most common population-based optimization methods can be expressed naturally within this formalism. The formalization even makes it possible to compare methods directly in a mathematical setting, as was done in Theorem 4.2.3, where the $(1+1)$–ES was shown to be the norm limit of Simulated Annealing in Euclidean space.

Chapters 5 and 7 exhibited the power of the formalization as a vehicle for mathematical analysis by proving the exact conditions under which optimizers are continuous. As shown in those chapters, genetic algorithms, evolution strategies, swarm optimizers, differential evolution, and stochastic gradient descent are continuous in most circumstances, especially on trajectories of unambiguous fitness. This continuity even carries over to performance criterion, so that the performance of most popular methods changes continuously along with the objective or the optimizer.

In Chapter 14, several extensions of these results were proposed. First, many of the same results should also apply to stochastic or dynamic objectives. Also, there are other ways that the space of optimizers could be formally analyzed, for example, by adding a base measure and considering only those optimizers that are defined with respect to that measure. The problem of convex control was also suggested as a way to leverage the best aspects of existing optimization methods on different problems.

15.1.2 Significance of the NFL Identification Theorems

In Chapters 9 and 10, the question of optimizer performance was studied in the context of the NFL theorems for optimization, which have been used to suggest that general-purpose learning is impossible. Perhaps surprisingly, No Free Lunch holds in arbitrary topological spaces subject to certain conditions. But the nature of these conditions are somewhat unreasonable, and under general assumptions of compressibility there exist general-purpose optimization methods.

Over the past decade, it has become a fundamental assumption within machine learning research that general-purpose learners do not exist, since NFL implies that every optimization method performs equivalently when averaged over all problems. In this light, general-purpose algorithms have come to be viewed with suspicion and distrust, primarily due to the truism that a good problem-specific solution will always outperform a general approach.

The exact nature of the conditions for NFL casts some doubt on this point of view. As the NFL Identification Theorem in Chapter 9 proved, a random test procedure for optimization methods produces an NFL result if and only if the corresponding function prior is independent and identically distributed at any finite collection of points. That is, NFL only holds if it is impossible to make any general assumptions about the nature of the universe on the basis of any finite set of observations.

The principle of Occam's razor, equally revered with NFL in machine learning research, suggests to the contrary that simple solutions should be more likely than complex solutions. This line of thinking was explored in Chapter 10, where it was suggested that with respect to general-purpose optimization, either Occam's Razor or NFL must hold. Both cannot be true at the same time. Occam's razor implies compressibility, and compressibility of any form prevents the objective value of solutions from being uncorrelated. Occam's razor is the older of the two concepts, and the more useful. NFL was found to imply that the world is unlearnable, a claim that is contradicted by the fact that humans frequently make useful predictions based on past events. As the study of the Optimization Game in Chapter 10 suggests, the idea that real-world problems are subject to NFL is almost tantamount to assuming the existence of a malevolent intelligence that purposely prevents learning. As a consequence, it is reasonable to assume that general-purpose optimization is possible.

The existence of general-purpose optimizers in no way implies that they outperform problem-specific optimizers. If the optimal solutions to a problem are known, then the optimizer that produces the known solutions will clearly perform best on a problem. It is often stated that specific solutions perform best, but this statement overlooks the effort expended to locate such solutions. In fact, the search for specific solutions to specific problem classes may be regarded as a general-purpose optimization method in which the human researcher applies his own native learning abilities as the primary tool. The very fact that researchers often succeed at finding successful problem-specific methods supports the hypothesis that strong general-purpose learners exist, and one should search for such learners by examining the techniques employed by successful researchers.

The discussion of linear projections in Chapter 10 suggests that for each problem class, represented as a specific function prior, there is some optimizer or subset of optimizers that perform optimally on it. A good general-purpose optimizer should perform well on a particular general-purpose function prior. It is an important line of future research to establish exactly what this means and how one might classify or describe function priors and the optimizers that perform well on them. An initial step in this direction was taken in Chapter 10 with the description of the information-maximizing optimization strategy that led to the development of evolutionary annealing. Still, there remains considerable theoretical work to be done in this regard. The characterization of NFL can be extended to information-restricted optimizers instead of just trajectory-restricted optimizers. It may even be possible to give conditions for NFL over the entire space of optimizers. Even so, NFL can potentially be avoided by a change of information function, a topic that deserves further study. Finally, the evidence strongly suggests that a prior that prefers shorter solutions over longer ones cannot be subject to NFL. This claim could be made more rigorous by presenting a formal proof.

15.1.3 Significance of Information Maximization

If indeed real-world problems are not subject to NFL, then what is the optimal optimizer? Chapter 10 introduced the *information-maximization principle* to answer this question: The optimal trajectory-restricted optimizer should be the one that makes the full use of the information obtained from function evaluations. The information-maximization principle led to the introduction of evolutionary annealing as a general-purpose optimization method that explicitly leverages all prior function evaluations. Evolutionary annealing was shown to converge asymptotically to the global optimum in Chapter 11. Its effectiveness was demonstrated experimentally in Chapter 12 for Euclidean space and Chapter 13 for neural networks, validating the information-maximization approach.

The information-maximization principle arises naturally by negating the requirements for an NFL prior. Since an NFL prior must be path independent, it stands to reason that the optimal optimizer for a non-NFL prior should maximally utilize whatever path dependencies are available. It would be of interest to derive a formal proof of this claim.

Information-maximization methods are inherently *martingale methods*, in contrast to the Markov Chain Monte Carlo (MCMC) methods that have figured prominently in statistical approaches to optimization [109, 147, 173, 221]. Where MCMC relies on the principle of detailed balance to guarantee that an iterated sample converges to some equilibrium distribution, martingale methods build a model of a desired random variable that is asymptotically correct by leveraging a source of increasing refined information. The Markov and martingale properties are the two most well-studied properties of stochastic processes. While the Markov property has been substantially exploited in artificial intelligence research, the martingale property has received less attention. The experimental results for evolutionary annealing in this dissertation suggest that martingale methods deserve further consideration for machine learning applications.

This dissertation introduced evolutionary annealing as a first example of an explicitly information-maximizing optimizer. The experimental results in Euclidean space in Chapter 12 demonstrated the effectiveness of this approach against a wide array of other optimization methods. The success of neuroannealing at learning complex neural networks in Chapter 13 made the case that evolutionary annealing is indeed a general-purpose optimization method.

There is still substantial room to improve evolutionary annealing. The methods for partitioning the search domain were designed to be simple to understand and implement. The quality of results might be improved by semantic partitioning. Additionally, the mutation distributions for evolutionary annealing determine how well evolutionary annealing will be aligned with a particular prior, and future work may discover a method of deriving mutation distributions to promote this alignment. Finally, it may be possible to derive a more compact information-maximizing optimizer by compressing previous evaluations so that the information obtained from previous evaluations is preserved without having to store or process all previously observed points.

The successful application of the information-maximization principle in this dissertation is a promising result. It is likely that this principle can be used to derive even more powerful optimizers in the future by more accurately mining evaluation information to reflect the nature of the prior governing real-world problems.

15.2 Final Thoughts

Optimization tasks are ubiquitous throughout the engineering disciplines. This observation is especially true of artificial intelligence and machine learning, where nearly every problem is expressed in terms of searching for a solution that is optimal according to some criterion. Thus the study of optimization is central to the quest for a strong artificial intelligence.

This dissertation studied the relationships among optimization methods at a general level by examining the probability distribution over the sequence of evaluation points produced by the optimization process. This study produced several results that perhaps seemed unintuitive at the outset. Optimizers are vectors. There is a well-defined objective measure of distance between any two optimizers. Between any two optimizers there is an entire spectrum of optimizers, and in most cases behavior and performance changes smoothly along this line. The discovery of these facts was made possible by analyzing the optimization process as a mathematical object.

The results presented only scratch the surface of what is possible to achieve using such an analysis. Future work on convex control of optimization portfolios could provide a way to allocate resources automatically to the best optimizer for a particular problem. The study of performance-based linear projections could even make it possible to analytically construct optimizers that are well-aligned with specific problem classes. Accurate approximations of optimizer methods may enable the implementation of near-optimal optimizers in cases where the theoretically optimal optimizer is uncomputable. Each of these topics can be explored by applied advanced mathematical theory to the space of optimizers as described in this dissertation.

The most important immediate contribution of this dissertation is the discovery of the No Free Lunch Identification Theorems, which proved that the only cases in which optimizer performance averaged over all problems is a constant consist of prior assumptions that objective evaluations at one point provide no information whatsoever about the value of the objective at any other point. Since it is patently absurd to claim that objective evaluations in real-world problems are completely uncorrelated, this theorem refutes the claim that no effective general-purpose optimization algorithms exist. A successful general-purpose optimizer should therefore structure its search so as to prioritize solutions that are more likely to be correct. In line with the principle of Occam's razor, simpler solutions should be preferred over more complex ones. For real-world problems, physical principles such a locality, smoothness, periodicity, and fractal structure should be used to guide the optimization process efficiently.

The construction of the diffusion prior proved mathematically that there exist very general problem classes on which certain optimization strategies outperform others. The recognition that the No Free Lunch theorems do not preclude general-purpose optimization led to the articulation of the information-maximization principle, which conjectures that the optimal optimization method for any particular problem class is the one that makes the full use of the information obtained from function evaluations. Explicitly information-maximizing optimizers form a new class of martingale-based optimization methods that deserve further theoretical and experimental study.

As an initial step in this direction, this dissertation proposed evolutionary annealing, which samples the same distribution as simulated annealing, but replacing the Markov-based Metropolis algorithm with a martingale representation that successively partitions the search domain. A proof of asymptotic global convergence stated the conditions under which evolutionary annealing can be expected to find the true global optimum. More importantly, experiments with real vectors and neural networks demonstrated the effectiveness of evolutionary annealing as a practical optimization method. Neuroannealing in particular was structured in such a way as to implement the information-maximization principle in accordance with Occam's razor, preferring simple network solutions over more complex ones until the simple networks have been ruled out.

Future research into information-maximizing optimizers will find more compact ways of partitioning the search space to represent the knowledge obtained through the optimization process. They will leverage the physical principles that govern the natural world in order to develop more effective means for choosing the next evaluation points. And they should ultimately outperform existing methods by a wide margin in more complex domains.

The greatest known optimizer at present is the human brain. By methodically applying scientific principles in conjunction with a creative instinct, human researchers have utilized knowledge gained from experience to construct increasingly refined and accurate models of the natural world. The exact nature of this creative instinct is poorly understood at present, but it may be surmised that human creativity comprises a set of hidden mental operations that project past observations into highly probable future states that accord with a core set of fundamental physical principles. If this hypothesis is true, then the most important endeavor in the search for a general artificial intelligence is to identify these fundamental principles. Once enumerated, these principles can be used to construct a general-purpose information-maximizing optimization method capable of human-level discoveries.

Thus the study of all optimization methods taken together has reinforced a fundamental insight regarding the nature of learning and artificial intelligence. General-purpose learners can be effective to the degree that their assumptions and biases reflect the physical laws of their environment. The study of artificial intelligence must in fact be a study of the abstract pillars of reality. By incorporating these principles, it should be possible to develop general-purpose learners of increasing capability and true intelligence.

Appendices

Appendix A

Performance Experiment Results

The appendix contains the results for the performance experiments in Chapters 8 and 12 in tabular form. Tables are presented for each of the performance criteria σ_ϵ^N (success probability with threshold ϵ), $\hat{\psi}_\epsilon^N$ (number of evaluations until success), ζ_{T_m} (average error after m evaluations), ϕ_1 (average error over all evaluations), and ϕ_2 (weighted average error over all evaluations) in order on the following pages (see Chapter 8 for details). Tables are grouped by the dimension of the experiments, with $d = 5$, 10, and 25. Additionally, the results for the neural network experiments in Chapter 13 are aggregated in a single chapter at the end of this appendix.

The values of ζ_{T_m}, ϕ_1, and ϕ_2 are scaled as described in Chapter 8. The scaling factors for each benchmark are listed in Table A.1. Additionally, variances are provided for ζ_{T_m}, ϕ_1, and ϕ_2 in separate tables so that statistical significance can be checked. All values are based on 200 trials. The variances for $\hat{\psi}_\epsilon^N$ were not given, since these averages were only computed for successful trials, and the number of successful trials varies in every case. The estimated values of σ_ϵ^N are accurate up to ± 0.005 with $p < 0.05$. Any values greater than $100,000$ appeared only in rare cases and were written simply as "∞".

Some of the algorithms were run with different parameters. The specific parameters are shown in the second column of the table. For CMA-ES and CMA-ES-R, the parameter is the population size. For DE, the parameters are given as CR / F, so that ".2/.9" means a crossover rate of .2 and a learning rate of .9. For PSO, the parameters are listed as ω/ϕ_g in the same way. The algorithms REA-P and REA-T were run with different learning rates as shown in Table 12.2, reproduced in this appendix as Table A.2 for convenience. The different results for each learning rates are shown order from top to bottom for each benchmark, matching the order of the values in Table 12.2 from left to

Table A.1: Scaling factors used for the scaled variants of ζ_{T_m}, ϕ_1, and ϕ_2 by dimension d.

	$d = 5$	$d = 10$	$d = 25$
sphere	1.247	12.552	84.481
ackley	2.442	4.059	5.509
log-ackley	0.393	24.250	184.941
whitley	43.998	20,726.140	5,018,118.903
shekel	10.472	10.247	–
rosenbrock	100.254	2,926.197	64,226.113
rastrigin	16.567	70.078	272.727
salomon	0.927	2.453	5.740
langerman	0.746	0.965	–
schwefel	99.070	187.058	270.419
griewank	5.125	43.479	290.635
weierstrass	3.248	10.275	34.156

right. That is, the smallest learning rates are given at the top, and the largest learning rates at the bottom. Extra entries in the table were marked with "–".

Table A.2: Learning rates η for REA-P and REA-T tested in the experiments. Lower values yield higher success probability at the cost of slower convergence.

	REA-P		REA-T		
Benchmark	$d = 5$	$d = 10$	$d = 5$	$d = 10$	$d = 25$
sphere	10	1, 10	10	1, 10	0.1, 1,10
ackley	0.25	0.25, 1	0.25	0.25, 1	0.05, 0.25, 1
log-ackley	0.25	0.25, 1	0.05, 0.25	0.25, 1	0.05, .25, 1
whitley	0.1	0.25, 1	0.05, 0.25	0.25, 1	0.05, 0.25, 1
shekel	0.1, 0.25	0.1, 1	0.1, 0.5, 1.0, 5.0	0.1, 1	–
rosenbrock	1	1, 5	5	1, 5	0.1, 1,5
rastrigin	0.01, 0.1	0.035, 1	0.01, 0.035, 0.050, 0.075	0.035, 1	0.01, 0.035, 1
salomon	2	1, 2	2	1, 2	0.1, 1, 2
langerman	0.1, 0.5	0.25, 1	0.1, 0.5, 1.0, 5.0	0.25, 1	–
schwefel	0.015	0.001, 0.01	0.001	0.001, 0.01	0.0001, 0.001, 0.01
griewank	1, 10	0.1, 1	0.025, 0.1, 0.25, 0.5	0.1, 1	0.01, 0.1, 1
weierstrass	5	1, 5	5	1, 5	0,1, 1, 5

Table A.3: Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 5 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.005	0.825	0.000	0.000	0.000	0.005	0.000	0.000
CG-R		1.000	0.000	0.000	0.005	0.995	1.000	0.010	1.000	0.990	0.180	0.000	0.000
CMA-ES	100	0.999	0.669	0.613	0.003	0.000	0.027	0.413	0.915	0.400	0.020	0.999	0.580
	750	1.000	1.000	0.890	0.625	0.000	0.300	1.000	1.000	0.860	0.110	1.000	0.755
	1250	1.000	1.000	0.875	0.750	0.000	0.905	1.000	1.000	0.900	0.200	1.000	0.695
	2500	1.000	1.000	0.820	0.245	0.000	0.795	1.000	1.000	0.900	0.345	1.000	0.255
CMA-ES-R	100	1.000	1.000	1.000	0.025	0.000	0.125	1.000	1.000	1.000	0.390	1.000	1.000
	750	1.000	0.905	0.235	0.600	0.000	0.000	0.980	1.000	0.510	0.080	0.080	0.030
	1250	1.000	0.000	0.010	0.605	0.000	0.000	0.670	1.000	0.015	0.020	0.000	0.000
	2500	0.730	0.000	0.000	0.064	0.000	0.000	0.175	0.064	0.020	0.025	0.000	0.000
DE	.2/.9	1.000	1.000	0.160	0.005	0.010	0.005	0.195	0.095	0.155	0.100	0.015	0.000
	.2/.2	1.000	1.000	0.925	0.175	0.110	0.135	0.920	1.000	0.305	0.785	1.000	0.485
	.9/.2	1.000	1.000	0.925	0.100	0.045	0.420	0.005	1.000	0.220	0.000	0.890	0.000
	.9/.9	0.170	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.069	0.085	0.775	0.005	0.064	0.020	0.420	0.000	0.095
GSS-R		1.000	0.480	0.025	0.650	0.755	0.740	0.095	0.771	0.054	1.000	0.085	0.680
NM		1.000	0.020	0.000	0.054	0.045	0.750	0.000	0.000	0.000	0.015	0.405	0.375
NM-R		1.000	0.185	0.940	1.000	1.000	1.000	0.175	0.135	0.040	0.855	1.000	1.000
PSO	–0.5/2	0.820	0.000	0.000	0.059	0.000	0.000	0.010	0.075	0.000	0.105	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	1.000	0.984	1.000	1.000	0.000	0.989	0.000	1.000	0.005	0.261	0.693	0.904
	R2	–	–	–	–	0.000	–	0.291	–	0.000	–	0.412	–
REA-T	R1	1.000	0.285	0.985	1.000	0.728	1.000	0.391	0.995	1.000	0.995	1.000	0.960
	R2	–	–	0.840	0.825	0.326	–	0.226	–	0.959	–	1.000	–
	R3	–	–	–	–	0.195	–	0.195	–	0.920	–	1.000	–
	R4	–	–	–	–	0.069	–	0.040	–	0.680	–	0.990	–
SA		1.000	0.000	0.000	0.000	0.000	1.000	0.115	0.190	0.125	0.000	0.000	0.000
rBOA		1.000	1.000	0.965	0.000	0.000	0.000	1.000	1.000	0.100	0.000	1.000	0.000
rGA		1.000	0.000	0.020	0.685	0.020	0.069	0.010	1.000	0.495	0.520	0.075	0.000

Table A.4: Results on performance criterion σ_ϵ^N with $\epsilon = 0.010$ and $N = 250,000$ in 5 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.005	0.825	0.000	0.000	0.000	0.005	0.000	0.000
CG-R		1.000	0.000	0.000	0.005	0.995	1.000	0.010	1.000	0.830	0.180	0.000	0.000
CMA-ES	100	0.999	0.665	0.613	0.003	0.000	0.004	0.413	0.000	0.209	0.020	0.678	0.265
	750	1.000	1.000	0.885	0.625	0.000	0.140	1.000	0.000	0.430	0.110	0.995	0.610
	1250	1.000	1.000	0.875	0.750	0.000	0.655	1.000	0.000	0.495	0.200	1.000	0.485
	2500	1.000	1.000	0.820	0.245	0.000	0.345	1.000	0.000	0.300	0.335	1.000	0.015
CMA-ES-R	100	1.000	1.000	1.000	0.025	0.000	0.020	1.000	0.000	0.995	0.380	1.000	0.995
	750	0.845	0.250	0.090	0.390	0.000	0.000	0.435	0.000	0.205	0.000	0.015	0.000
	1250	0.845	0.000	0.010	0.320	0.000	0.000	0.190	0.000	0.000	0.000	0.000	0.000
	2500	0.160	0.000	0.000	0.030	0.000	0.000	0.080	0.000	0.000	0.005	0.000	0.000
DE	.2/.9	0.995	0.660	0.005	0.000	0.000	0.000	0.000	0.000	0.049	0.000	0.000	0.000
	.2/.2	1.000	0.765	0.925	0.155	0.049	0.030	0.780	0.000	0.125	0.735	0.175	0.245
	.9/.2	1.000	0.580	0.875	0.075	0.040	0.120	0.005	0.000	0.015	0.000	0.000	0.000
	.9/.9	0.000	0.700	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.069	0.085	0.700	0.005	0.000	0.005	0.420	0.000	0.095
GSS-R		1.000	0.480	0.025	0.650	0.755	0.675	0.095	0.000	0.045	1.000	0.000	0.675
NM		1.000	0.020	0.000	0.054	0.045	0.750	0.000	0.000	0.000	0.015	0.005	0.355
NM-R		1.000	0.185	0.940	1.000	1.000	1.000	0.175	0.000	0.025	0.855	0.975	1.000
PSO	–0.5/2	0.160	0.000	0.000	0.049	0.000	0.000	0.000	0.015	0.000	0.105	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	1.000	0.045	0.572	1.000	0.000	0.989	0.000	0.000	0.000	0.000	0.015	0.000
	R2	–	–	–	–	0.000	–	0.010	–	0.000	–	0.000	–
REA-T	R1	1.000	0.285	0.985	1.000	0.728	1.000	0.391	0.000	0.889	0.995	0.326	0.885
	R2	–	–	0.840	0.825	0.326	–	0.226	–	0.688	–	0.095	–
	R3	–	–	–	–	0.195	–	0.195	–	0.580	–	0.090	–
	R4	–	–	–	–	0.069	–	0.040	–	0.300	–	0.040	–
SA		1.000	0.000	0.000	0.000	0.000	0.120	0.000	0.000	0.000	0.000	0.000	0.000
rBOA		1.000	1.000	0.010	0.000	0.000	0.000	1.000	0.855	0.000	0.000	1.000	0.000
rGA		1.000	0.000	0.020	0.255	0.000	0.000	0.000	0.000	0.245	0.010	0.000	0.000

Table A.5: Results on performance criterion σ_ϵ^N with $\epsilon = 0.001$ and $N = 250,000$ in 5 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.005	0.825	0.000	0.000	0.000	0.005	0.000	0.000
CG-R		1.000	0.000	0.000	0.005	0.995	1.000	0.010	1.000	0.820	0.170	0.000	0.000
CMA-ES	100	0.999	0.665	0.609	0.003	0.000	0.000	0.413	0.000	0.209	0.018	0.339	0.100
	750	1.000	1.000	0.885	0.625	0.000	0.069	1.000	0.000	0.430	0.105	0.970	0.460
	1250	1.000	1.000	0.875	0.750	0.000	0.490	1.000	0.000	0.495	0.200	0.985	0.365
	2500	1.000	1.000	0.820	0.245	0.000	0.145	1.000	0.000	0.295	0.290	1.000	0.000
CMA-ES-R	100	1.000	1.000	1.000	0.025	0.000	0.010	1.000	0.000	0.995	0.370	1.000	0.000
	750	0.190	0.000	0.015	0.145	0.000	0.000	0.030	0.000	0.085	0.000	0.005	0.000
	1250	0.415	0.000	0.000	0.064	0.000	0.000	0.025	0.000	0.000	0.000	0.000	0.000
	2500	0.025	0.000	0.000	0.010	0.000	0.000	0.059	0.000	0.000	0.000	0.000	0.000
DE	.2/.9	0.425	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.010	0.000	0.000	0.000
	.2/.2	1.000	0.005	0.925	0.150	0.040	0.000	0.645	0.000	0.010	0.705	0.005	0.185
	.9/.2	1.000	0.000	0.660	0.059	0.035	0.015	0.005	0.000	0.000	0.000	0.000	0.000
	.9/.9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.069	0.085	0.660	0.005	0.000	0.005	0.420	0.000	0.095
GSS-R		1.000	0.480	0.025	0.650	0.755	0.635	0.095	0.000	0.045	1.000	0.000	0.675
NM		1.000	0.020	0.000	0.054	0.045	0.750	0.000	0.000	0.000	0.015	0.000	0.350
NM-R		1.000	0.185	0.940	1.000	1.000	1.000	0.175	0.000	0.025	0.855	0.430	1.000
PSO	–0.5/2	0.020	0.000	0.000	0.035	0.000	0.000	0.000	0.005	0.000	0.105	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	1.000	0.000	0.000	1.000	0.000	0.135	0.000	0.000	0.000	0.000	0.000	0.000
	R2	–	–	–	–	0.000	–	0.000	–	0.000	–	0.000	–
REA-T	R1	1.000	0.285	0.985	1.000	0.728	0.995	0.391	0.000	0.889	0.995	0.045	0.845
	R2	–	–	0.840	0.825	0.326	–	0.226	–	0.688	–	0.020	–
	R3	–	–	–	–	0.195	–	0.195	–	0.580	–	0.005	–
	R4	–	–	–	–	0.069	–	0.040	–	0.300	–	0.010	–
SA		0.100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
rBOA		1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.850	0.000	0.000	1.000	0.000
rGA		1.000	0.000	0.010	0.015	0.000	0.000	0.000	0.000	0.240	0.000	0.000	0.000

Table A.6: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.100$ and $N = 250,000$ in 5 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	3.0	16.6	–	–	–	1.0	–	–
CG-R		0.0	–	–	873.0	545.6	21.0	1995.5	78.2	532.5	1138.0	–	–
CMA-ES	100	4.9	21.8	21.5	19.0	–	224.3	18.0	12.8	14.8	107.0	18.2	64.8
	750	87.9	149.6	151.0	197.0	–	1127.4	169.5	102.6	135.6	356.3	190.9	390.0
	1250	204.9	339.5	347.6	413.5	–	1135.6	373.8	224.9	287.2	703.4	417.8	835.2
	2500	614.8	1071.6	1078.6	1235.2	–	2207.2	1146.7	741.1	868.4	1752.8	1324.1	2185.7
CMA-ES-R	100	4.9	28.8	37.5	1540.6	–	1322.6	52.7	20.1	100.1	1165.1	18.2	71.7
	750	200.6	915.7	1006.5	1174.8	–	–	690.6	104.5	1178.9	1476.2	1440.2	1524.8
	1250	409.1	–	318.5	1395.8	–	–	1174.3	261.6	1916.3	1749.7	–	–
	2500	1268.3	–	–	1828.8	–	–	1515.0	1021.1	1743.7	1920.0	–	–
DE	.2/.9	225.6	355.7	1786.7	1138.0	2306.5	1656.0	1896.3	1687.1	1415.5	1874.9	1840.0	–
	.2/.2	65.4	311.7	331.4	486.2	1265.6	834.9	1051.2	571.3	1315.0	1204.8	383.3	1592.2
	.9/.2	68.3	338.6	1067.5	1080.0	1647.2	1611.9	1612.0	874.9	1269.4	–	1285.6	–
	.9/.9	1249.6	351.7	–	–	–	–	–	–	–	–	–	–
GSS		5.0	–	–	11.5	11.1	339.3	11.0	10.4	6.7	10.8	–	8.5
GSS-R		4.9	1165.1	1078.7	932.2	1064.3	348.5	1144.0	951.7	203.6	235.7	1227.6	609.0
NM		0.4	1.2	–	1.2	1.0	3.9	–	–	–	1.6	1.4	0.0
NM-R		0.4	951.1	697.9	190.2	137.4	8.0	1028.9	1171.2	1314.7	798.8	10.7	5.9
PSO	–0.5/2	705.2	–	–	1144.0	–	–	1242.5	854.1	–	137.5	–	0.0
	1/2	–	–	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	8.6	1284.3	146.0	150.5	–	140.1	–	255.3	1865.0	1182.3	68.7	118.7
	R2	–	–	–	–	–	–	1113.3	–	–	–	1495.2	–
REA-T	R1	7.7	104.2	138.7	124.3	229.1	100.1	515.7	26.3	63.3	1133.2	395.3	19.9
	R2	–	–	67.9	63.4	101.3	–	257.1	–	34.6	–	199.6	–
	R3	–	–	–	–	76.0	–	148.3	–	27.5	–	132.8	–
	R4	–	–	–	–	42.8	–	56.6	–	16.8	–	99.2	–
SA		6.7	–	–	–	–	237.3	1369.0	1456.2	1182.8	–	–	–
rBOA		7.7	37.5	753.6	–	–	–	36.9	23.6	1349.6	–	15.2	–
rGA		26.1	–	37.2	840.8	135.5	411.2	251.5	320.7	1113.1	620.0	1531.6	–

Table A.7: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.010$ and $N = 250,000$ in 5 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	3.0	16.6	–	–	–	1.0	–	–
CG-R		0.0	–	–	873.0	548.0	21.0	1995.5	206.4	914.6	1138.0	–	–
CMA-ES	100	7.3	25.3	24.5	20.0	–	159.5	20.7	–	17.6	110.2	24.0	95.2
	750	126.9	187.2	185.2	211.9	–	1288.6	207.2	–	162.9	419.0	230.5	502.4
	1250	296.3	435.6	419.6	437.4	–	1447.1	462.0	–	345.2	785.3	508.5	1055.0
	2500	951.3	1349.8	1296.7	1287.7	–	2364.1	1436.7	–	1060.4	2036.9	1530.3	2283.3
CMA-ES-R	100	7.3	32.6	40.3	1542.8	–	974.0	55.4	–	298.3	1168.4	53.2	421.2
	750	914.5	1112.4	1386.0	1199.4	–	–	1109.0	–	1227.7	–	1527.3	–
	1250	1111.6	–	349.5	1447.9	–	–	1177.0	–	–	–	–	–
	2500	1714.8	–	–	1783.3	–	–	1575.0	–	–	1825.0	–	–
DE	.2/.9	939.4	1487.0	2436.0	–	–	–	–	–	1698.1	–	–	–
	.2/.2	126.0	1459.5	492.1	617.7	1367.1	862.1	1470.5	–	1728.8	1399.5	1630.2	1750.7
	.9/.2	167.7	1516.2	1455.8	1409.8	1719.0	1811.7	1740.0	–	1396.3	–	–	–
	.9/.9	–	1550.5	–	–	–	–	–	–	–	–	–	–
GSS		7.6	–	–	13.3	14.4	330.6	14.0	–	7.0	13.4	–	13.7
GSS-R		7.6	1168.1	1080.5	934.4	1066.9	331.8	1146.6	–	233.1	238.2	–	623.5
NM		0.7	1.2	–	1.3	1.2	4.5	–	–	–	2.0	2.0	0.0
NM-R		0.7	951.5	698.3	190.3	137.7	8.6	1029.1	–	1278.6	799.0	565.3	6.5
PSO	–0.5/2	1087.0	–	–	1149.1	–	–	–	1511.3	–	257.4	–	0.0
	1/2	–	–	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	17.2	1971.0	1423.9	246.6	–	385.4	–	–	–	–	357.6	–
	R2	–	–	–	–	–	–	1707.0	–	–	–	–	–
REA-T	R1	11.8	114.7	156.4	133.4	264.8	351.9	553.3	–	80.6	1287.3	541.3	28.9
	R2	–	–	77.4	68.3	117.0	–	274.9	–	41.7	–	251.6	–
	R3	–	–	–	–	88.0	–	160.8	–	32.2	–	160.4	–
	R4	–	–	–	–	50.7	–	62.3	–	19.1	–	118.1	–
SA		165.4	–	–	–	–	1439.0	–	–	–	–	–	–
rBOA		10.8	44.0	812.0	–	–	–	39.3	659.5	–	–	51.7	–
rGA		34.9	–	78.7	1278.5	–	–	–	–	1127.6	1877.0	–	–

Table A.8: Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 0.001$ and $N = 250,000$ in 5 dimensions.

$\frac{1}{100}\hat{\psi}_\epsilon^N$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	3.0	16.6	–	–	–	1.0	–	–
CG-R		0.0	–	–	873.0	548.0	21.0	1995.5	206.4	969.5	1170.5	–	–
CMA-ES	100	9.7	29.2	26.1	21.0	–	–	23.0	–	20.2	42.7	35.9	177.4
	750	164.3	243.3	223.4	230.3	–	1619.7	244.8	–	193.1	411.5	266.7	617.1
	1250	386.6	561.8	506.1	471.0	–	1573.0	553.6	–	400.5	881.9	592.0	1275.1
	2500	1244.5	1785.6	1555.9	1392.3	–	2400.8	1755.0	–	1288.1	2294.8	1805.0	–
CMA-ES-R	100	9.7	36.3	42.9	1545.6	–	1012.5	57.9	–	300.9	1203.4	131.1	–
	750	1163.3	–	1694.6	1205.7	–	–	1588.5	–	1392.2	–	1095.0	–
	1250	1393.7	–	–	1574.7	–	–	1275.0	–	–	–	–	–
	2500	1960.0	–	–	1787.5	–	–	1791.6	–	–	–	–	–
DE	.2/.9	1831.0	–	–	–	–	–	–	–	1932.5	–	–	–
	.2/.2	193.1	1661.0	639.9	685.5	1309.2	–	1658.9	–	1767.5	1513.9	2499.0	1884.6
	.9/.2	273.1	–	1543.2	1507.7	1696.1	1979.6	1862.0	–	–	–	–	–
	.9/.9	–	–	–	–	–	–	–	–	–	–	–	–
GSS		10.3	–	–	15.7	16.8	373.3	16.0	–	10.0	16.0	–	18.5
GSS-R		10.2	1172.1	1084.0	936.4	1069.6	349.5	1149.8	–	234.8	240.9	–	628.5
NM		1.0	1.5	–	1.5	1.6	4.9	–	–	–	2.3	–	0.0
NM-R		1.0	951.9	698.5	190.6	137.9	9.0	1029.5	–	1278.8	799.3	1118.1	6.6
PSO	–0.5/2	1905.5	–	–	1237.4	–	–	–	1993.0	–	676.1	–	0.0
	1/2	–	–	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	166.2	–	–	848.6	–	1691.5	–	–	–	–	–	–
	R2	–	–	–	–	–	–	–	–	–	–	–	–
REA-T	R1	15.8	126.6	174.1	141.7	298.9	838.1	585.3	–	92.6	1447.6	527.1	37.7
	R2	–	–	86.4	72.5	131.6	–	290.4	–	48.1	–	253.2	–
	R3	–	–	–	–	98.8	–	170.4	–	37.7	–	149.0	–
	R4	–	–	–	–	56.2	–	68.2	–	23.4	–	119.5	–
SA		1311.0	–	–	–	–	–	–	–	–	–	–	–
rBOA		13.2	47.9	–	–	–	–	42.2	661.1	–	–	59.0	–
rGA		94.8	–	387.0	1592.0	–	–	–	–	1110.6	–	–	–

Table A.9: Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 5 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	2.323	168.967	14.178	0.789	1.069	2.555	3.831	1.290	2.298	13.171	3.038
CG-R		0.000	0.918	38.969	0.307	0.416	0.000	0.401	0.035	0.630	0.676	1.536	1.659
CMA-ES	100	0.000	0.026	1.110	0.089	0.762	0.012	0.046	0.112	0.261	0.978	0.001	0.060
	750	0.000	0.000	0.289	0.016	0.748	0.015	0.000	0.106	0.070	0.653	0.000	0.555
	1250	0.036	0.390	10.236	0.312	0.839	0.062	0.246	0.108	0.422	1.187	0.166	1.305
	2500	0.254	0.675	26.277	0.469	0.858	0.229	0.565	0.498	0.802	1.051	0.349	1.113
CMA-ES-R	100	0.000	0.000	0.013	0.038	0.736	0.012	0.001	0.107	0.041	0.564	0.000	0.006
	750	0.057	0.387	26.086	0.333	0.865	0.061	0.035	0.107	0.711	0.652	0.282	0.424
	1250	0.123	0.693	20.884	0.367	0.865	0.128	0.247	0.166	0.787	1.068	0.389	1.090
	2500	0.264	0.685	27.786	0.485	0.864	0.227	0.549	0.519	0.915	1.051	0.383	1.130
DE	.2/.9	0.075	0.057	12.595	0.354	0.794	0.096	0.250	0.454	0.678	0.234	0.242	0.494
	.2/.2	0.000	0.052	1.432	0.103	0.693	0.018	0.151	0.194	0.645	0.277	0.034	0.461
	.9/.2	0.001	0.055	7.202	0.215	0.702	0.024	0.395	0.212	0.600	0.807	0.073	0.951
	.9/.9	0.383	0.055	34.254	0.525	0.856	0.338	0.690	0.666	0.795	0.918	0.528	1.000
GSS		0.000	2.356	18.674	0.103	0.718	0.005	0.312	0.478	0.872	0.309	0.098	0.755
GSS-R		0.000	0.128	12.708	0.062	0.666	0.005	0.222	0.297	0.806	0.123	0.074	0.499
NM		0.000	0.212	77.499	0.239	0.745	0.040	1.643	2.728	1.281	1.482	1.398	0.291
NM-R		0.000	0.118	3.471	0.010	0.125	0.000	0.162	0.460	1.005	0.279	0.003	0.000
PSO	–0.5/2	0.232	0.338	37.586	0.476	0.892	0.334	1.195	0.620	1.145	0.427	0.396	0.000
	1/2	4.590	0.340	91.646	132.687	0.937	9.609	2.163	2.018	1.289	0.897	4.086	0.000
REA-P	R1	0.000	0.219	0.226	0.000	0.863	0.002	0.625	0.109	0.858	0.108	0.016	0.064
	R2	–	–	–	–	0.866	–	0.087	–	0.867	–	0.049	–
REA-T	R1	0.000	0.064	0.037	0.000	0.180	0.000	0.360	0.108	0.005	0.601	0.034	0.010
	R2	–	–	0.402	0.003	0.450	–	0.075	–	0.030	–	0.006	–
	R3	–	–	–	–	0.547	–	0.067	–	0.053	–	0.006	–
	R4	–	–	–	–	0.660	–	0.119	–	0.169	–	0.008	–
SA		0.005	0.815	26.249	0.326	0.862	0.003	0.141	0.311	0.618	0.681	0.108	0.646
rBOA		0.000	0.000	0.435	0.194	0.811	0.024	0.000	0.065	0.609	2.046	0.000	1.127
rGA		0.000	0.435	16.841	0.066	0.756	0.015	0.216	0.108	0.941	0.136	0.046	0.533

Table A.10: Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 5 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.412	58.034	141.565	0.079	10.672	1.106	1.283	0.045	0.853	7.080	0.562
CG-R		0.000	0.208	14.311	0.093	0.319	0.000	0.169	0.061	0.522	0.266	0.909	0.231
CMA-ES	100	0.000	0.038	1.483	0.085	0.037	0.006	0.046	0.020	0.227	0.443	0.001	0.086
	750	0.000	0.000	0.817	0.022	0.026	0.003	0.000	0.006	0.120	0.342	0.000	0.304
	1250	0.017	0.071	3.803	0.040	0.039	0.020	0.078	0.011	0.160	0.217	0.031	0.172
	2500	0.114	0.089	5.905	0.060	0.046	0.101	0.132	0.107	0.155	0.179	0.084	0.153
CMA-ES-R	100	0.000	0.000	0.195	0.042	0.008	0.005	0.010	–	0.082	0.345	0.000	0.008
	750	0.051	0.218	8.272	0.128	0.023	0.049	0.036	0.001	0.174	0.331	0.106	0.309
	1250	0.104	0.090	7.908	0.074	0.021	0.057	0.086	0.133	0.164	0.218	0.090	0.172
	2500	0.115	0.091	5.727	0.061	0.025	0.098	0.141	0.108	0.161	0.214	0.087	0.159
DE	.2/.9	0.051	0.033	5.104	0.071	0.059	0.054	0.096	0.127	0.143	0.136	0.052	0.132
	.2/.2	0.000	0.031	1.209	0.052	0.143	0.010	0.068	0.062	0.156	0.183	0.011	0.113
	.9/.2	0.001	0.033	2.787	0.057	0.123	0.009	0.117	0.063	0.106	0.238	0.018	0.164
	.9/.9	0.174	0.032	6.976	0.083	0.030	0.169	0.150	0.159	0.152	0.200	0.149	0.142
GSS		0.000	0.415	10.806	0.071	0.224	0.008	0.186	0.335	0.296	0.378	0.039	0.514
GSS-R		0.000	0.046	6.740	0.052	0.251	0.007	0.109	0.163	0.326	0.198	0.028	0.458
NM		0.000	0.103	49.791	0.181	0.168	0.361	0.958	1.213	0.071	0.619	4.643	0.397
NM-R		0.000	0.046	3.318	0.019	0.259	0.000	0.081	0.168	0.252	0.198	0.001	0.000
PSO	–0.5/2	0.253	0.066	11.900	0.159	0.022	0.476	0.400	0.290	0.150	0.225	0.225	0.000
	1/2	2.090	0.070	19.158	245.385	0.014	7.659	0.454	0.473	0.022	0.070	1.876	0.000
REA-P	R1	0.000	0.045	0.492	0.008	0.022	0.009	0.146	0.008	0.169	0.129	0.010	0.130
	R2	–	–	–	–	0.026	–	0.044	–	0.167	–	0.011	–
REA-T	R1	0.000	0.043	0.305	0.000	0.291	0.000	0.092	0.007	0.018	0.183	0.008	0.057
	R2	–	–	0.922	0.009	0.313	–	0.040	–	0.085	–	0.003	–
	R3	–	–	–	–	0.271	–	0.044	–	0.116	–	0.003	–
	R4	–	–	–	–	0.184	–	0.057	–	0.217	–	0.004	–
SA		0.002	0.112	6.018	0.048	0.026	0.010	0.063	0.103	0.146	0.237	0.023	0.113
rBOA		0.000	0.000	0.236	0.014	0.024	0.006	0.000	0.047	0.128	0.361	0.000	0.158
rGA		0.000	0.150	10.581	0.053	0.121	0.014	0.116	0.001	0.329	0.167	0.010	0.077

Table A.11: Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 5 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	2.323	168.967	14.178	0.789	1.069	2.555	3.831	1.290	2.298	13.171	3.038
CG-R		0.000	0.690	24.186	0.208	0.118	0.000	0.225	0.001	0.153	0.418	0.626	1.416
CMA-ES	100	0.000	0.026	1.101	0.086	0.762	0.012	0.046	0.112	0.261	0.969	0.001	0.056
	750	0.000	0.000	0.288	0.013	0.748	0.005	0.000	0.106	0.069	0.392	0.000	0.044
	1250	0.000	0.000	0.333	0.008	0.758	0.001	0.000	0.105	0.045	0.271	0.000	0.066
	2500	0.006	0.150	2.185	0.188	0.781	0.031	0.027	0.104	0.078	0.811	0.083	0.942
CMA-ES-R	100	0.000	0.000	0.000	0.014	0.735	0.006	0.000	0.107	0.001	0.293	0.000	0.002
	750	0.012	0.104	17.128	0.168	0.840	0.027	0.004	0.106	0.444	0.156	0.173	0.124
	1250	0.022	0.598	17.158	0.163	0.841	0.053	0.016	0.105	0.591	0.281	0.300	0.219
	2500	0.123	0.581	21.118	0.353	0.837	0.065	0.091	0.383	0.623	0.807	0.300	0.846
DE	.2/.9	0.008	0.011	3.346	0.189	0.654	0.030	0.074	0.287	0.537	0.037	0.131	0.239
	.2/.2	0.000	0.011	0.190	0.018	0.616	0.009	0.018	0.109	0.485	0.050	0.007	0.136
	.9/.2	0.000	0.011	0.571	0.081	0.608	0.004	0.228	0.108	0.444	0.450	0.021	0.686
	.9/.9	0.213	0.012	25.813	0.442	0.823	0.178	0.510	0.494	0.642	0.676	0.369	0.842
GSS		0.000	2.356	18.674	0.103	0.718	0.003	0.312	0.478	0.872	0.309	0.098	0.755
GSS-R		0.000	0.074	6.954	0.021	0.449	0.004	0.123	0.177	0.790	0.002	0.054	0.287
NM		0.000	0.212	77.499	0.239	0.745	0.040	1.643	2.728	1.281	1.482	1.398	0.291
NM-R		0.000	0.088	0.812	0.000	0.000	0.000	0.084	0.302	0.744	0.101	0.001	0.000
PSO	–0.5/2	0.091	0.338	29.919	0.392	0.888	0.128	0.798	0.403	1.082	0.320	0.267	0.000
	1/2	4.586	0.340	91.117	132.687	0.937	9.609	2.163	2.018	1.289	0.893	4.064	0.000
REA-P	R1	0.000	0.077	0.047	0.000	0.829	0.001	0.435	0.107	0.699	0.082	0.016	0.053
	R2	–	–	–	–	0.837	–	0.056	–	0.700	–	0.029	–
REA-T	R1	0.000	0.064	0.037	0.000	0.178	0.000	0.038	0.108	0.005	0.009	0.003	0.010
	R2	–	–	0.402	0.003	0.450	–	0.055	–	0.030	–	0.004	–
	R3	–	–	–	–	0.547	–	0.067	–	0.053	–	0.006	–
	R4	–	–	–	–	0.660	–	0.119	–	0.169	–	0.008	–
SA		0.002	0.691	19.443	0.275	0.828	0.000	0.058	0.169	0.478	0.391	0.078	0.500
rBOA		0.000	0.000	0.213	0.175	0.790	0.019	0.000	0.031	0.454	2.046	0.000	0.938
rGA		0.000	0.434	16.837	0.034	0.756	0.005	0.214	0.107	0.582	0.135	0.033	0.459

Table A.12: Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 5 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.412	58.034	141.565	0.079	10.672	1.106	1.283	0.045	0.853	7.080	0.562
CG-R		0.000	0.159	9.245	0.068	0.252	0.000	0.098	0.010	0.285	0.213	0.424	0.194
CMA-ES	100	0.000	0.038	1.482	0.084	0.037	0.006	0.046	0.020	0.226	0.443	0.001	0.085
	750	0.000	0.000	0.817	0.018	0.026	0.004	0.000	0.006	0.120	0.251	0.000	0.106
	1250	0.000	0.000	0.906	0.016	0.036	0.001	0.000	0.010	0.087	0.202	0.000	0.107
	2500	0.003	0.074	1.783	0.049	0.054	0.008	0.019	0.011	0.066	0.177	0.019	0.221
CMA-ES-R	100	0.000	0.000	0.000	0.011	0.003	0.003	0.000	0.004	0.005	0.180	0.000	0.000
	750	0.013	0.155	8.885	0.153	0.024	0.009	0.005	0.005	0.265	0.136	0.069	0.050
	1250	0.031	0.081	6.749	0.120	0.028	0.019	0.016	0.009	0.094	0.212	0.055	0.093
	2500	0.071	0.084	4.677	0.077	0.029	0.025	0.073	0.107	0.098	0.190	0.050	0.310
DE	.2/.9	0.007	0.008	2.240	0.069	0.189	0.016	0.044	0.085	0.155	0.035	0.046	0.068
	.2/.2	0.000	0.007	0.666	0.015	0.253	0.007	0.024	0.015	0.189	0.096	0.003	0.063
	.9/.2	0.000	0.007	0.800	0.045	0.234	0.004	0.078	0.007	0.177	0.211	0.005	0.117
	.9/.9	0.090	0.009	5.714	0.059	0.039	0.083	0.109	0.116	0.084	0.181	0.083	0.119
GSS		0.000	0.415	10.806	0.071	0.224	0.007	0.186	0.335	0.296	0.378	0.039	0.514
GSS-R		0.000	0.039	3.283	0.028	0.334	0.007	0.051	0.062	0.331	0.023	0.021	0.452
NM		0.000	0.103	49.791	0.181	0.168	0.361	0.958	1.213	0.071	0.619	4.643	0.397
NM-R		0.000	0.034	1.264	0.002	0.000	0.000	0.044	0.095	0.225	0.123	0.000	0.000
PSO	–0.5/2	0.081	0.066	9.366	0.141	0.021	0.100	0.386	0.229	0.166	0.180	0.085	0.000
	1/2	2.092	0.070	19.117	245.385	0.014	7.659	0.454	0.473	0.022	0.075	1.898	0.000
REA-P	R1	0.000	0.039	0.020	0.000	0.036	0.007	0.104	0.000	0.123	0.121	0.010	0.128
	R2	–	–	–	–	0.046	–	0.037	–	0.104	–	0.007	–
REA-T	R1	0.000	0.043	0.305	0.000	0.292	0.000	0.032	0.007	0.018	0.017	0.001	0.057
	R2	–	–	0.922	0.009	0.313	–	0.036	–	0.085	–	0.002	–
	R3	–	–	–	–	0.271	–	0.044	–	0.116	–	0.003	–
	R4	–	–	–	–	0.184	–	0.057	–	0.217	–	0.004	–
SA		0.001	0.094	4.782	0.043	0.040	0.000	0.043	0.050	0.166	0.196	0.018	0.082
rBOA		0.000	0.000	0.090	0.021	0.022	0.007	0.000	0.046	0.167	0.361	0.000	0.133
rGA		0.000	0.150	10.581	0.032	0.122	0.004	0.116	0.000	0.450	0.167	0.007	0.060

Table A.13: Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 5 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	2.323	168.967	14.178	0.789	1.069	2.555	3.831	1.290	2.298	13.171	3.038
CG-R		0.000	0.572	17.835	0.154	0.003	0.000	0.158	0.000	0.016	0.278	0.379	1.268
CMA-ES	100	0.000	0.026	1.101	0.085	0.762	0.012	0.046	0.112	0.261	0.969	0.001	0.055
	750	0.000	0.000	0.288	0.013	0.748	0.004	0.000	0.106	0.069	0.392	0.000	0.044
	1250	0.000	0.000	0.333	0.008	0.758	0.000	0.000	0.105	0.045	0.271	0.000	0.044
	2500	0.000	0.000	0.473	0.023	0.755	0.000	0.000	0.103	0.046	0.158	0.000	0.084
CMA-ES-R	100	0.000	0.000	0.000	0.008	0.732	0.003	0.000	0.107	0.000	0.154	0.000	0.001
	750	0.004	0.015	10.552	0.050	0.823	0.021	0.001	0.104	0.268	0.035	0.129	0.091
	1250	0.003	0.529	14.654	0.049	0.824	0.038	0.005	0.104	0.517	0.048	0.259	0.149
	2500	0.056	0.513	17.045	0.236	0.819	0.034	0.033	0.326	0.526	0.228	0.254	0.228
DE	.2/.9	0.001	0.003	0.831	0.096	0.514	0.020	0.017	0.215	0.441	0.005	0.065	0.144
	.2/.2	0.000	0.003	0.189	0.011	0.606	0.008	0.005	0.107	0.340	0.035	0.003	0.040
	.9/.2	0.000	0.003	0.128	0.046	0.581	0.002	0.144	0.107	0.259	0.275	0.013	0.548
	.9/.9	0.143	0.003	21.453	0.405	0.795	0.119	0.424	0.416	0.584	0.557	0.303	0.750
GSS		0.000	2.356	18.674	0.103	0.718	0.002	0.312	0.478	0.872	0.309	0.098	0.755
GSS-R		0.000	0.043	4.809	0.006	0.215	0.001	0.090	0.138	0.768	0.000	0.039	0.239
NM		0.000	0.212	77.499	0.239	0.745	0.040	1.643	2.728	1.281	1.482	1.398	0.291
NM-R		0.000	0.067	0.148	0.000	0.000	0.000	0.057	0.237	0.585	0.033	0.000	0.000
PSO	–0.5/2	0.044	0.338	24.066	0.331	0.883	0.080	0.475	0.268	1.000	0.286	0.220	0.000
	1/2	4.586	0.340	91.059	132.687	0.937	9.609	2.163	2.018	1.289	0.893	4.064	0.000
REA-P	R1	0.000	0.012	0.023	0.000	0.777	0.000	0.329	0.107	0.620	0.073	0.016	0.050
	R2	–	–	–	–	0.811	–	0.049	–	0.638	–	0.020	–
REA-T	R1	0.000	0.064	0.037	0.000	0.178	0.000	0.038	0.108	0.005	0.001	0.003	0.010
	R2	–	–	0.402	0.003	0.450	–	0.055	–	0.030	–	0.004	–
	R3	–	–	–	–	0.547	–	0.067	–	0.053	–	0.006	–
	R4	–	–	–	–	0.660	–	0.119	–	0.169	–	0.008	–
SA		0.001	0.605	15.293	0.244	0.802	0.000	0.026	0.125	0.339	0.267	0.065	0.430
rBOA		0.000	0.000	0.140	0.164	0.778	0.017	0.000	0.012	0.314	2.046	0.000	0.840
rGA		0.000	0.433	16.835	0.015	0.756	0.003	0.213	0.107	0.389	0.135	0.028	0.408

Table A.14: Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 5 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.412	58.034	141.565	0.079	10.672	1.106	1.283	0.045	0.853	7.080	0.562
CG-R		0.000	0.137	7.385	0.063	0.046	0.000	0.072	0.000	0.065	0.176	0.240	0.172
CMA-ES	100	0.000	0.038	1.482	0.083	0.037	0.006	0.046	0.020	0.226	0.444	0.001	0.085
	750	0.000	0.000	0.817	0.018	0.026	0.003	0.000	0.006	0.120	0.251	0.000	0.106
	1250	0.000	0.000	0.906	0.016	0.036	0.001	0.000	0.010	0.087	0.202	0.000	0.091
	2500	0.000	0.000	1.049	0.016	0.053	0.001	0.000	0.013	0.067	0.135	0.000	0.077
CMA-ES-R	100	0.000	0.000	0.000	0.007	0.021	0.002	0.000	0.004	0.002	0.132	0.000	0.000
	750	0.004	0.019	8.982	0.096	0.039	0.004	0.001	0.012	0.258	0.064	0.060	0.032
	1250	0.006	0.075	5.660	0.088	0.030	0.012	0.006	0.011	0.130	0.070	0.046	0.045
	2500	0.045	0.081	4.417	0.104	0.032	0.012	0.031	0.087	0.128	0.168	0.043	0.106
DE	.2/.9	0.001	0.002	0.706	0.049	0.286	0.011	0.015	0.071	0.206	0.004	0.032	0.044
	.2/.2	0.000	0.002	0.666	0.011	0.267	0.006	0.017	0.000	0.231	0.087	0.001	0.039
	.9/.2	0.000	0.002	0.522	0.032	0.271	0.004	0.058	0.000	0.152	0.174	0.004	0.105
	.9/.9	0.062	0.002	5.050	0.051	0.050	0.048	0.101	0.097	0.091	0.151	0.060	0.122
GSS		0.000	0.415	10.806	0.071	0.224	0.006	0.186	0.335	0.296	0.378	0.039	0.514
GSS-R		0.000	0.038	1.887	0.012	0.314	0.004	0.044	0.048	0.330	0.000	0.014	0.460
NM		0.000	0.103	49.791	0.181	0.168	0.361	0.958	1.213	0.071	0.619	4.643	0.397
NM-R		0.000	0.034	0.588	0.000	0.000	0.000	0.034	0.072	0.169	0.081	0.000	0.000
PSO	–0.5/2	0.046	0.066	7.733	0.148	0.021	0.055	0.289	0.157	0.198	0.187	0.055	0.000
	1/2	2.092	0.070	19.026	245.385	0.014	7.659	0.454	0.473	0.022	0.075	1.898	0.000
REA-P	R1	0.000	0.009	0.010	0.000	0.112	0.003	0.075	0.000	0.090	0.114	0.010	0.127
	R2	–	–	–	–	0.050	–	0.035	–	0.075	–	0.004	–
REA-T	R1	0.000	0.043	0.305	0.000	0.292	0.000	0.032	0.007	0.018	0.016	0.001	0.057
	R2	–	–	0.922	0.009	0.313	–	0.036	–	0.085	–	0.002	–
	R3	–	–	–	–	0.271	–	0.044	–	0.116	–	0.003	–
	R4	–	–	–	–	0.184	–	0.057	–	0.217	–	0.004	–
SA		0.000	0.089	4.097	0.040	0.061	0.000	0.026	0.034	0.170	0.159	0.014	0.067
rBOA		0.000	0.000	0.062	0.024	0.024	0.007	0.000	0.032	0.149	0.361	0.000	0.104
rGA		0.000	0.150	10.581	0.023	0.122	0.001	0.116	0.000	0.462	0.167	0.006	0.061

Table A.15: Results on performance criterion ϕ_1 (scaled) in 5 dimensions.

ϕ_1 (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	2.323	168.968	19.012	0.789	1.381	2.555	3.831	1.290	2.298	13.172	3.038
CG-R		0.000	0.711	26.518	17.445	0.145	0.005	0.251	0.021	0.218	0.450	0.839	1.429
CMA-ES	100	0.000	0.027	1.111	0.087	0.762	0.012	0.047	0.112	0.261	0.974	0.001	0.060
	750	0.016	0.030	1.468	0.039	0.755	0.022	0.029	0.123	0.109	0.482	0.019	0.175
	1250	0.033	0.073	3.142	0.066	0.773	0.036	0.062	0.142	0.141	0.450	0.043	0.314
	2500	0.067	0.221	8.189	0.180	0.788	0.072	0.165	0.205	0.279	0.543	0.117	0.677
CMA-ES-R	100	0.000	0.000	0.032	0.018	0.735	0.007	0.001	0.107	0.016	0.319	0.000	0.008
	750	0.032	0.144	17.035	0.169	0.841	0.045	0.036	0.122	0.464	0.259	0.191	0.213
	1250	0.056	0.600	17.902	0.174	0.841	0.076	0.075	0.147	0.619	0.364	0.311	0.355
	2500	0.138	0.582	21.621	0.348	0.838	0.101	0.197	0.399	0.662	0.619	0.307	0.696
DE	.2/.9	0.036	0.021	5.220	0.207	0.644	0.058	0.106	0.309	0.552	0.087	0.147	0.278
	.2/.2	0.011	0.019	1.354	0.053	0.634	0.023	0.055	0.143	0.484	0.109	0.027	0.187
	.9/.2	0.009	0.020	2.601	0.106	0.623	0.017	0.248	0.141	0.428	0.480	0.041	0.710
	.9/.9	0.243	0.021	26.708	0.465	0.823	0.207	0.527	0.513	0.663	0.699	0.395	0.855
GSS		0.000	2.356	18.674	0.103	0.718	0.003	0.312	0.478	0.872	0.309	0.098	0.755
GSS-R		0.000	0.172	7.178	0.023	0.391	0.004	0.128	0.184	0.784	0.031	0.051	0.308
NM		0.000	0.212	77.499	0.239	0.745	0.040	1.643	2.728	1.281	1.482	1.398	0.291
NM-R		0.000	0.088	1.538	0.004	0.034	0.000	0.099	0.327	0.751	0.132	0.002	0.000
PSO	–0.5/2	0.107	0.338	29.792	0.398	0.887	0.162	0.782	0.412	1.071	0.341	0.279	0.000
	1/2	4.590	0.340	91.196	132.701	0.937	9.609	2.163	2.018	1.289	0.894	4.069	0.000
REA-P	R1	0.000	0.089	0.226	0.008	0.822	0.001	0.456	0.113	0.712	0.093	0.018	0.057
	R2	–	–	–	–	0.835	–	0.073	–	0.721	–	0.032	–
REA-T	R1	0.000	0.075	0.721	0.011	0.213	0.000	0.111	0.109	0.018	0.148	0.018	0.010
	R2	–	–	0.683	0.008	0.458	–	0.093	–	0.035	–	0.011	–
	R3	–	–	–	–	0.552	–	0.090	–	0.056	–	0.010	–
	R4	–	–	–	–	0.662	–	0.127	–	0.170	–	0.010	–
SA		0.003	0.697	19.848	0.278	0.828	0.001	0.077	0.201	0.472	0.420	0.081	0.518
rBOA		0.000	0.002	0.318	0.177	0.792	0.019	0.003	0.034	0.456	2.046	0.000	0.958
rGA		0.003	0.435	16.944	0.046	0.757	0.013	0.216	0.115	0.596	0.137	0.037	0.464

Table A.16: Variance for performance criterion ϕ_1 (scaled) in 5 dimensions.

$Var(\phi_1)$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.412	58.034	141.302	0.079	10.644	1.106	1.283	0.045	0.853	7.080	0.562
CG-R		0.000	0.123	7.287	23.847	0.142	0.004	0.074	0.016	0.184	0.160	0.319	0.148
CMA-ES	100	0.000	0.038	1.480	0.083	0.037	0.006	0.046	0.020	0.226	0.441	0.001	0.084
	750	0.005	0.002	0.795	0.018	0.025	0.007	0.005	0.006	0.114	0.229	0.003	0.100
	1250	0.009	0.006	0.900	0.015	0.032	0.011	0.010	0.012	0.079	0.164	0.006	0.090
	2500	0.021	0.019	1.279	0.018	0.048	0.019	0.025	0.022	0.054	0.104	0.015	0.112
CMA-ES-R	100	0.000	0.001	0.041	0.010	0.006	0.002	0.001	0.003	0.015	0.142	0.000	0.003
	750	0.010	0.086	7.053	0.098	0.025	0.011	0.007	0.006	0.198	0.088	0.055	0.048
	1250	0.024	0.063	5.714	0.074	0.022	0.015	0.014	0.017	0.083	0.109	0.042	0.075
	2500	0.049	0.065	3.610	0.057	0.023	0.024	0.038	0.078	0.080	0.127	0.040	0.130
DE	.2/.9	0.013	0.007	1.554	0.046	0.172	0.017	0.031	0.064	0.138	0.031	0.034	0.050
	.2/.2	0.004	0.007	0.656	0.017	0.228	0.008	0.021	0.011	0.162	0.091	0.004	0.044
	.9/.2	0.003	0.007	0.749	0.034	0.214	0.006	0.057	0.010	0.123	0.168	0.004	0.080
	.9/.9	0.068	0.008	4.486	0.049	0.031	0.064	0.082	0.090	0.070	0.135	0.063	0.099
GSS		0.000	0.415	10.806	0.071	0.224	0.006	0.186	0.335	0.296	0.378	0.039	0.514
GSS-R		0.000	0.034	2.410	0.019	0.250	0.007	0.043	0.051	0.321	0.034	0.014	0.434
NM		0.000	0.103	49.791	0.181	0.168	0.361	0.958	1.213	0.071	0.619	4.643	0.397
NM-R		0.000	0.030	1.143	0.004	0.037	0.000	0.034	0.072	0.159	0.089	0.003	0.000
PSO	–0.5/2	0.077	0.066	7.646	0.128	0.020	0.122	0.275	0.178	0.160	0.174	0.080	0.000
	1/2	2.089	0.070	18.984	245.378	0.014	7.659	0.454	0.473	0.022	0.072	1.890	0.000
REA-P	R1	0.000	0.019	0.128	0.002	0.044	0.005	0.072	0.002	0.095	0.114	0.010	0.128
	R2	–	–	–	–	0.038	–	0.034	–	0.081	–	0.004	–
REA-T	R1	0.000	0.042	0.308	0.001	0.275	0.000	0.030	0.007	0.018	0.026	0.002	0.057
	R2	–	–	0.918	0.010	0.306	–	0.034	–	0.084	–	0.002	–
	R3	–	–	–	–	0.266	–	0.043	–	0.116	–	0.003	–
	R4	–	–	–	–	0.182	–	0.057	–	0.216	–	0.004	–
SA		0.001	0.074	3.719	0.035	0.032	0.002	0.030	0.041	0.127	0.154	0.013	0.061
rBOA		0.000	0.001	0.088	0.018	0.020	0.006	0.002	0.035	0.120	0.361	0.000	0.097
rGA		0.003	0.149	10.561	0.030	0.121	0.009	0.116	0.003	0.388	0.167	0.006	0.047

Table A.17: Results on performance criterion ϕ_2 (scaled) in 5 dimensions.

ϕ_2 (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	2.323	169.502	1226.341	0.789	104.733	2.555	3.831	1.290	2.481	13.448	3.038
CG-R		0.000	1.700	109.497	3534.461	0.735	1.584	1.356	1.307	1.218	2.213	10.130	2.480
CMA-ES	100	0.000	0.242	5.513	0.234	0.779	0.032	0.392	0.124	0.534	1.806	0.060	1.030
	750	0.927	0.958	46.539	0.928	0.912	0.967	1.031	0.917	1.239	1.889	0.852	1.687
	1250	0.908	0.984	47.301	0.921	0.914	0.895	1.025	0.917	1.254	1.879	0.879	1.678
	2500	0.623	0.903	42.169	0.653	0.897	0.604	0.905	0.805	1.198	1.701	0.709	1.416
CMA-ES-R	100	0.000	0.241	5.719	0.231	0.784	0.032	0.368	0.125	0.517	1.811	0.059	1.019
	750	0.840	0.939	45.881	0.777	0.904	0.810	0.985	0.873	1.163	1.774	0.858	1.518
	1250	0.802	0.958	45.670	0.799	0.906	0.757	0.965	0.883	1.206	1.841	0.808	1.484
	2500	0.602	0.894	41.902	0.662	0.898	0.641	0.855	0.791	1.192	1.731	0.688	1.440
DE	.2/.9	1.541	0.187	57.456	2.877	0.904	1.873	1.202	1.199	1.187	1.324	1.469	1.324
	.2/.2	1.426	0.198	54.959	2.977	0.900	1.687	1.200	1.245	1.174	1.341	1.372	1.361
	.9/.2	1.086	0.196	53.513	1.643	0.896	1.299	1.201	1.078	1.137	1.500	1.101	1.535
	.9/.9	1.504	0.195	59.798	2.669	0.904	1.646	1.294	1.222	1.170	1.531	1.431	1.434
GSS		0.000	2.357	18.888	0.142	0.720	0.035	0.317	0.478	0.873	0.310	0.118	0.761
GSS-R		0.000	2.289	17.773	0.140	0.747	0.048	0.311	0.455	0.852	0.348	0.115	0.716
NM		0.000	0.212	77.499	0.239	0.745	0.040	1.643	2.728	1.281	1.482	1.398	0.291
NM-R		0.000	0.189	41.857	0.152	0.673	0.009	0.926	1.661	1.263	1.094	0.106	0.028
PSO	−0.5/2	0.381	0.338	42.878	0.651	0.894	0.650	1.404	0.957	1.222	0.965	0.514	0.000
	1/2	4.648	0.340	92.256	133.918	0.937	9.682	2.168	2.019	1.289	0.933	4.128	0.000
REA-P	R1	0.117	0.346	20.960	0.361	0.905	0.156	1.288	0.569	1.161	1.117	0.563	0.511
	R2	–	–	–	–	0.906	–	0.957	–	1.172	–	0.185	–
REA-T	R1	0.014	0.815	50.302	1.174	0.905	0.066	1.371	0.447	1.031	1.519	1.277	0.271
	R2	–	–	37.816	0.523	0.893	–	1.230	–	0.838	–	0.747	–
	R3	–	–	–	–	0.886	–	1.139	–	0.740	–	0.466	–
	R4	–	–	–	–	0.864	–	0.976	–	0.516	–	0.333	–
SA		0.049	1.180	47.480	0.596	0.906	0.051	1.399	1.506	1.162	1.424	0.179	1.089
rBOA		0.028	0.306	9.952	0.324	0.880	0.074	0.573	0.202	1.177	2.046	0.024	1.556
rGA		0.865	0.855	40.262	2.884	0.889	1.281	0.618	0.906	1.245	0.554	0.804	0.963

Table A.18: Variance for performance criterion ϕ_2 (scaled) in 5 dimensions.

$Var(\phi_2)$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.412	58.596	874.804	0.079	37.801	1.106	1.283	0.045	1.006	7.060	0.562
CG-R		0.000	0.345	52.084	1924.897	0.067	0.405	0.598	1.001	0.239	1.056	5.079	0.363
CMA-ES	100	0.000	0.050	2.643	0.057	0.035	0.005	0.136	0.027	0.153	0.260	0.016	0.260
	750	0.375	0.128	9.138	0.475	0.014	0.498	0.263	0.194	0.084	0.222	0.324	0.178
	1250	0.347	0.113	8.048	0.447	0.008	0.468	0.217	0.182	0.063	0.244	0.295	0.182
	2500	0.260	0.107	8.226	0.165	0.017	0.288	0.214	0.173	0.117	0.265	0.235	0.211
CMA-ES-R	100	0.000	0.055	2.715	0.059	0.035	0.004	0.149	0.026	0.163	0.278	0.016	0.276
	750	0.375	0.124	8.994	0.278	0.015	0.399	0.214	0.169	0.136	0.249	0.280	0.214
	1250	0.304	0.115	7.661	0.307	0.012	0.382	0.226	0.170	0.106	0.219	0.260	0.213
	2500	0.261	0.136	8.427	0.188	0.018	0.301	0.221	0.175	0.117	0.231	0.222	0.209
DE	.2/.9	0.771	0.045	11.097	3.650	0.016	1.116	0.305	0.282	0.115	0.287	0.577	0.168
	.2/.2	0.678	0.049	11.159	3.536	0.027	1.055	0.305	0.296	0.125	0.313	0.537	0.204
	.9/.2	0.536	0.048	12.044	1.920	0.023	1.019	0.272	0.243	0.147	0.293	0.500	0.173
	.9/.9	0.673	0.047	10.784	2.955	0.017	0.908	0.266	0.241	0.113	0.270	0.522	0.182
GSS		0.000	0.416	10.780	0.075	0.221	0.247	0.185	0.334	0.296	0.378	0.039	0.513
GSS-R		0.000	0.481	9.274	0.078	0.181	0.328	0.164	0.369	0.309	0.382	0.040	0.474
NM		0.000	0.103	49.791	0.181	0.168	0.361	0.958	1.213	0.071	0.619	4.643	0.397
NM-R		0.000	0.085	35.131	0.093	0.212	0.046	0.612	0.813	0.097	0.426	1.194	0.076
PSO	−0.5/2	0.399	0.066	13.468	0.760	0.022	0.737	0.368	0.300	0.113	0.152	0.320	0.000
	1/2	2.082	0.070	18.365	245.263	0.014	7.609	0.453	0.474	0.022	0.092	1.847	0.000
REA-P	R1	0.411	0.078	5.882	0.044	0.015	0.185	0.276	0.147	0.123	0.312	0.256	0.302
	R2	–	–	–	–	0.015	–	0.230	–	0.128	–	0.035	–
REA-T	R1	0.009	0.105	9.325	0.834	0.014	0.026	0.263	0.103	0.167	0.241	0.454	0.089
	R2	–	–	6.246	0.073	0.021	–	0.280	–	0.133	–	0.228	–
	R3	–	–	–	–	0.016	–	0.237	–	0.125	–	0.119	–
	R4	–	–	–	–	0.021	–	0.216	–	0.146	–	0.069	–
SA		0.031	0.160	10.611	0.280	0.014	0.160	0.584	0.446	0.149	0.273	0.035	0.172
rBOA		0.048	0.126	4.964	0.064	0.024	0.082	0.342	0.079	0.117	0.361	0.013	0.204
rGA		0.612	0.181	12.626	4.793	0.024	1.041	0.193	0.267	0.089	0.228	0.450	0.140

Table A.19: Results on performance criterion σ_ϵ^N with $\epsilon = 1.000$ and $N = 250,000$ in 10 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.000	0.870	0.000	0.045	1.000	0.000	0.145	0.000
CG-R		1.000	0.000	0.000	0.000	0.910	1.000	0.000	1.000	1.000	0.000	1.000	0.000
CMA-ES	100	1.000	1.000	0.230	0.000	0.000	0.000	0.360	1.000	1.000	0.000	1.000	0.069
	750	1.000	1.000	0.820	0.005	0.000	0.000	1.000	1.000	1.000	0.005	1.000	0.600
	1250	1.000	1.000	0.945	0.054	0.000	0.000	1.000	1.000	1.000	0.000	1.000	0.805
	2500	1.000	1.000	0.980	0.095	0.000	0.000	1.000	1.000	1.000	0.000	1.000	0.960
CMA-ES-R	100	1.000	1.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000	0.000	1.000	1.000
	750	1.000	1.000	1.000	0.000	0.000	0.000	1.000	1.000	1.000	0.000	1.000	0.000
	1250	1.000	1.000	0.740	0.000	0.000	0.000	0.980	1.000	1.000	0.000	1.000	0.000
	2500	0.995	1.000	0.085	0.000	0.000	0.000	0.090	1.000	1.000	0.000	0.195	0.000
DE	.2/.9	0.999	1.000	0.000	0.000	0.000	0.000	0.000	0.995	1.000	0.011	0.010	0.000
	.2/.2	1.000	1.000	0.510	0.059	0.010	0.064	0.365	1.000	1.000	0.260	1.000	0.840
	.9/.2	1.000	1.000	0.220	0.015	0.030	0.035	0.000	1.000	1.000	0.000	1.000	0.000
	.9/.9	0.005	1.000	0.000	0.000	0.000	0.000	0.000	0.015	1.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.049	0.030	0.730	0.000	0.775	1.000	0.385	0.995	0.075
GSS-R		1.000	1.000	0.000	0.205	0.125	0.725	0.000	1.000	1.000	0.985	1.000	0.505
NM		1.000	0.985	0.000	0.000	0.015	0.750	0.000	0.005	1.000	0.000	0.875	0.005
NM-R		1.000	1.000	0.005	0.105	0.940	1.000	0.000	0.245	1.000	0.000	1.000	0.905
PSO	–0.5/2	1.000	0.015	0.000	0.000	0.000	0.000	0.025	0.985	1.000	0.005	0.000	1.000
	1/2	0.000	0.029	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000
REA-P	R1	1.000	0.000	0.085	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000	1.000
	R2	1.000	0.000	0.005	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.830	0.255
REA-T	R1	1.000	0.885	0.320	0.059	0.095	0.175	0.015	1.000	1.000	0.000	1.000	0.855
	R2	1.000	1.000	0.235	0.135	0.125	0.355	0.000	1.000	1.000	0.000	1.000	1.000
SA		1.000	0.000	0.000	0.000	0.000	0.840	0.000	1.000	1.000	0.000	0.015	0.000
rBOA		1.000	1.000	0.135	0.000	0.000	0.000	0.999	1.000	1.000	0.000	1.000	0.000
rGA		1.000	0.045	0.000	0.005	0.005	0.000	0.000	1.000	1.000	0.030	0.935	0.000

Table A.20: Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 10 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.000	0.870	0.000	0.005	0.000	0.000	0.000	0.000
CG-R		1.000	0.000	0.000	0.000	0.910	1.000	0.000	1.000	0.005	0.000	0.100	0.000
CMA-ES	100	1.000	0.220	0.190	0.000	0.000	0.000	0.110	0.080	0.010	0.000	1.000	0.000
	750	1.000	0.995	0.780	0.005	0.000	0.000	0.980	1.000	0.110	0.005	1.000	0.220
	1250	1.000	1.000	0.935	0.054	0.000	0.000	1.000	1.000	0.085	0.000	1.000	0.580
	2500	1.000	1.000	0.975	0.090	0.000	0.000	1.000	1.000	0.205	0.000	1.000	0.815
CMA-ES-R	100	1.000	1.000	0.995	0.000	0.000	0.000	0.995	0.985	0.290	0.000	1.000	0.265
	750	1.000	0.015	0.005	0.000	0.000	0.000	0.000	1.000	0.595	0.000	0.000	0.000
	1250	1.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.470	0.000	0.000	0.000
	2500	0.645	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000
DE	.2/.9	0.801	0.580	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	.2/.2	1.000	0.490	0.435	0.059	0.005	0.020	0.125	0.170	0.000	0.240	0.950	0.290
	.9/.2	1.000	0.455	0.160	0.015	0.030	0.000	0.000	0.040	0.000	0.000	0.049	0.000
	.9/.9	0.000	0.490	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.045	0.030	0.575	0.000	0.000	0.005	0.385	0.240	0.035
GSS-R		1.000	0.000	0.000	0.155	0.125	0.555	0.000	0.005	0.000	0.985	0.890	0.315
NM		1.000	0.000	0.000	0.000	0.015	0.750	0.000	0.000	0.000	0.000	0.565	0.000
NM-R		1.000	0.000	0.005	0.105	0.940	1.000	0.000	0.000	0.000	0.000	1.000	0.215
PSO	–0.5/2	0.460	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R2	1.000	0.000	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
REA-T	R1	1.000	0.000	0.064	0.059	0.095	0.000	0.000	1.000	0.635	0.000	0.295	0.800
	R2	1.000	0.000	0.170	0.115	0.125	0.000	0.000	1.000	0.410	0.000	0.000	1.000
SA		1.000	0.000	0.000	0.000	0.000	0.005	0.000	0.000	0.000	0.000	0.000	0.000
rBOA		1.000	1.000	0.000	0.000	0.000	0.000	0.975	1.000	0.000	0.000	1.000	0.000
rGA		1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.005	0.000	0.000	0.000	0.000

Table A.21: Results on performance criterion σ_ϵ^N with $\epsilon = 0.010$ and $N = 250,000$ in 10 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.000	0.870	0.000	0.000	0.000	0.000	0.000	0.000
CG-R		1.000	0.000	0.000	0.000	0.910	1.000	0.000	0.960	0.000	0.000	0.000	0.000
CMA-ES	100	1.000	0.220	0.190	0.000	0.000	0.000	0.110	0.000	0.005	0.000	0.995	0.000
	750	1.000	0.995	0.780	0.005	0.000	0.000	0.980	0.000	0.054	0.005	1.000	0.035
	1250	1.000	1.000	0.935	0.054	0.000	0.000	1.000	0.000	0.015	0.000	1.000	0.390
	2500	1.000	1.000	0.975	0.085	0.000	0.000	1.000	0.000	0.000	0.000	0.995	0.495
CMA-ES-R	100	1.000	1.000	0.995	0.000	0.000	0.000	0.995	0.000	0.225	0.000	1.000	0.000
	750	0.730	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.430	0.000	0.000	0.000
	1250	0.120	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.140	0.000	0.000	0.000
	2500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DE	.2/.9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	.2/.2	1.000	0.000	0.435	0.045	0.005	0.005	0.105	0.000	0.000	0.220	0.005	0.069
	.9/.2	1.000	0.000	0.120	0.010	0.030	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	.9/.9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.045	0.030	0.550	0.000	0.000	0.000	0.385	0.010	0.035
GSS-R		1.000	0.000	0.000	0.155	0.125	0.540	0.000	0.000	0.000	0.985	0.135	0.315
NM		1.000	0.000	0.000	0.000	0.015	0.750	0.000	0.000	0.000	0.000	0.054	0.000
NM-R		1.000	0.000	0.005	0.105	0.940	1.000	0.000	0.000	0.000	0.000	0.980	0.110
PSO	–0.5/2	0.035	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R2	0.975	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
REA-T	R1	1.000	0.000	0.010	0.059	0.095	0.000	0.000	0.000	0.345	0.000	0.000	0.740
	R2	1.000	0.000	0.170	0.095	0.125	0.000	0.000	0.000	0.235	0.000	0.000	1.000
SA		0.020	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
rBOA		1.000	0.995	0.000	0.000	0.000	0.000	0.955	0.010	0.000	0.000	1.000	0.000
rGA		1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.22: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 1.000$ and $N = 250,000$ in 10 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	–	53.6	–	1.2	0.0	–	57.3	–
CG-R		0.0	–	–	–	750.1	64.5	–	57.9	0.0	–	267.4	–
CMA-ES	100	7.5	18.2	29.0	–	–	–	42.0	7.3	0.0	–	14.9	67.5
	750	58.8	98.8	147.8	187.0	–	–	160.4	56.4	0.0	405.0	84.7	402.5
	1250	127.9	186.1	267.2	337.2	–	–	287.7	125.5	0.0	–	161.3	704.9
	2500	433.8	478.0	650.0	986.8	–	–	768.3	372.1	0.0	–	614.8	1684.7
CMA-ES-R	100	7.3	17.9	165.7	–	–	–	109.2	7.0	0.0	–	14.8	113.3
	750	57.8	100.6	193.3	–	–	–	173.5	54.9	0.0	–	83.3	–
	1250	125.9	183.1	725.2	–	–	–	744.5	123.8	0.0	–	162.0	–
	2500	656.1	493.3	814.7	–	–	–	1602.7	387.2	0.0	–	1314.1	–
DE	.2/.9	367.9	8.7	–	–	–	–	–	690.6	0.0	2239.5	1800.0	–
	.2/.2	98.6	9.1	597.4	639.3	1394.5	894.1	1838.8	152.1	0.0	1523.1	280.3	1655.2
	.9/.2	93.2	9.7	1717.6	1888.0	993.8	1644.4	–	116.4	0.0	–	406.7	–
	.9/.9	1752.0	8.5	–	–	–	–	–	1694.0	0.0	–	–	–
GSS		12.8	–	–	46.0	33.1	275.6	–	8.1	0.0	27.3	31.2	15.1
GSS-R		13.4	353.9	–	1157.2	1294.7	231.4	–	66.2	0.0	496.7	33.0	1036.6
NM		3.4	1.2	–	–	9.3	35.4	–	2.0	0.0	–	4.0	2.0
NM-R		3.5	6.8	1055.0	1030.0	756.3	45.7	–	1156.5	0.0	–	11.7	875.4
PSO	–0.5/2	417.1	5.3	–	–	–	–	1288.0	627.1	0.0	554.0	–	0.0
	1/2	–	1.0	–	–	–	–	–	–	0.0	–	–	0.0
REA-P	R1	114.5	–	1539.6	–	–	–	–	343.1	0.0	–	–	1070.2
	R2	203.7	–	792.0	–	–	–	–	156.0	0.0	–	1515.3	139.7
REA-T	R1	105.6	1849.5	2352.0	1425.8	750.8	2258.1	2303.0	133.6	0.0	–	481.3	60.5
	R2	166.7	1282.1	1443.6	2099.3	917.0	2200.4	–	155.1	0.0	–	562.4	105.9
SA		3.8	–	–	–	–	458.7	–	254.0	0.0	–	1755.3	–
rBOA		13.8	44.1	1470.0	–	–	–	326.6	2.9	0.2	–	1.7	–
rGA		38.7	89.2	–	2456.0	825.0	–	–	42.9	0.0	1112.1	769.8	–

Table A.23: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.100$ and $N = 250,000$ in 10 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	–	53.6	–	1.0	–	–	–	–
CG-R		0.0	–	–	–	760.0	64.5	–	250.8	246.0	–	1592.7	–
CMA-ES	100	11.7	25.0	32.6	–	–	–	30.5	22.3	26.0	–	24.5	–
	750	74.6	152.6	174.7	202.0	–	–	179.6	134.7	186.5	450.0	137.7	529.0
	1250	149.5	271.5	312.0	362.2	–	–	313.8	243.0	335.0	–	244.9	901.8
	2500	528.0	670.7	760.2	1080.5	–	–	948.3	606.7	766.4	–	842.8	2043.8
CMA-ES-R	100	11.4	144.7	202.5	–	–	–	401.0	666.4	1140.3	–	24.4	1084.4
	750	73.6	1329.6	1207.0	–	–	–	–	133.4	1065.9	–	–	–
	1250	149.2	–	–	–	–	–	–	242.5	1210.5	–	–	–
	2500	1243.6	–	–	–	–	–	–	604.2	–	–	–	–
DE	.2/.9	1687.7	1732.1	–	–	–	–	–	–	–	–	–	–
	.2/.2	200.1	1713.0	799.1	793.9	1227.0	1250.7	1943.6	1934.5	–	1726.6	1382.4	2059.4
	.9/.2	257.2	1700.3	1885.5	1968.0	1188.0	–	–	1892.8	–	–	1898.1	–
	.9/.9	–	1810.0	–	–	–	–	–	–	–	–	–	–
GSS		22.0	–	–	48.4	43.0	299.6	–	–	20.0	36.5	48.9	23.4
GSS-R		22.8	–	–	1237.4	1304.0	278.4	–	1872.0	–	506.0	743.8	1125.0
NM		4.3	–	–	–	10.0	41.3	–	–	–	–	7.4	–
NM-R		4.4	–	1058.0	1031.0	757.3	52.3	–	–	–	–	34.3	1269.5
PSO	–0.5/2	1194.1	–	–	–	–	–	–	–	–	–	–	0.0
	1/2	–	–	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	1217.1	–	–	–	–	–	–	–	–	–	–	–
	R2	315.2	–	1601.0	–	–	–	–	–	–	–	–	–
REA-T	R1	215.4	–	2423.9	1543.2	942.4	–	–	660.4	2184.0	–	2310.2	95.9
	R2	346.1	–	1542.9	2169.0	1123.8	–	–	819.7	968.0	–	–	164.4
SA		115.5	–	–	–	–	1347.0	–	–	–	–	–	–
rBOA		22.5	275.3	–	–	–	–	347.5	94.2	–	–	99.8	–
rGA		59.9	–	–	–	–	–	–	2157.0	–	–	–	–

Table A.24: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.010$ and $N = 250,000$ in 10 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.0	–	–	–	–	53.6	–	–	–	–	–	–
CG-R		0.0	–	–	–	767.9	64.5	–	592.0	–	–	–	–
CMA-ES	100	15.7	29.7	36.8	–	–	–	34.2	–	33.0	–	28.5	–
	750	90.1	173.8	195.8	217.0	–	–	197.4	–	244.5	480.0	155.3	678.0
	1250	171.2	301.3	351.8	388.5	–	–	338.5	–	491.3	–	275.4	1117.6
	2500	685.5	852.6	939.1	1113.2	–	–	1104.8	–	–	–	1028.6	2347.7
CMA-ES-R	100	15.4	149.4	206.4	–	–	–	404.8	–	1054.5	–	29.0	–
	750	1042.4	–	–	–	–	–	–	–	1118.7	–	–	–
	1250	856.4	–	–	–	–	–	–	–	1402.4	–	–	–
	2500	–	–	–	–	–	–	–	–	–	–	–	–
DE	.2/.9	–	–	–	–	–	–	–	–	–	–	–	–
	.2/.2	305.7	–	975.8	802.6	1324.0	1415.0	2153.1	–	–	1817.9	2262.0	2252.8
	.9/.2	428.2	–	1961.5	1822.5	1357.8	–	–	–	–	–	–	–
	.9/.9	–	–	–	–	–	–	–	–	–	–	–	–
GSS		31.3	–	–	57.3	52.0	433.5	–	–	–	45.6	61.0	33.4
GSS-R		31.9	–	–	1246.0	1313.1	429.1	–	–	–	515.4	1041.6	1113.5
NM		5.2	–	–	–	11.3	44.3	–	–	–	–	9.6	–
NM-R		5.1	–	1060.0	1032.0	758.3	55.4	–	–	–	–	564.5	1352.5
PSO	–0.5/2	1307.8	–	–	–	–	–	–	–	–	–	–	0.0
	1/2	–	–	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	–	–	–	–	–	–	–	–	–	–	–	–
	R2	1342.4	–	–	–	–	–	–	–	–	–	–	–
REA-T	R1	330.7	–	2398.0	1655.2	1119.4	–	–	–	2297.5	–	–	132.7
	R2	513.6	–	1696.7	2253.8	1333.2	–	–	–	1076.3	–	–	221.8
SA		969.2	–	–	–	–	–	–	–	–	–	–	–
rBOA		30.8	309.0	–	–	–	–	368.7	905.0	–	–	167.5	–
rGA		546.3	–	–	–	–	–	–	–	–	–	–	–

Table A.25: Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 10 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	1.431	6.279	0.133	0.846	0.014	1.267	2.045	1.000	1.329	0.143	1.940
CG-R		0.000	0.936	3.325	0.003	0.598	0.045	0.485	0.069	0.995	0.707	0.022	1.335
CMA-ES	100	0.000	0.032	0.067	0.002	0.851	0.001	0.029	0.073	0.624	0.898	0.000	0.288
	750	0.000	0.000	0.010	0.002	0.852	0.002	0.000	0.040	0.403	1.070	0.000	0.733
	1250	0.000	0.142	0.145	0.002	0.878	0.002	0.260	0.041	0.602	1.171	0.002	1.276
	2500	0.351	0.689	2.029	0.032	0.967	0.199	0.681	0.545	0.999	1.063	0.283	1.066
CMA-ES-R	100	0.000	0.006	0.013	0.001	0.851	0.001	0.009	0.055	0.425	0.652	0.000	0.048
	750	0.001	0.085	0.031	0.002	0.880	0.003	0.009	0.040	0.371	0.958	0.017	0.517
	1250	0.002	0.142	0.142	0.002	0.964	0.003	0.246	0.040	0.597	1.114	0.019	1.120
	2500	0.365	0.687	2.072	0.033	0.967	0.204	0.669	0.544	0.999	1.084	0.292	1.072
DE	.2/.9	0.127	0.105	1.337	0.008	0.959	0.082	0.357	0.527	0.985	0.444	0.147	0.587
	.2/.2	0.003	0.108	0.394	0.003	0.924	0.010	0.257	0.278	0.987	0.454	0.025	0.580
	.9/.2	0.009	0.107	0.978	0.004	0.913	0.010	0.563	0.257	0.951	0.924	0.032	1.026
	.9/.9	0.539	0.107	2.646	0.125	0.966	0.369	0.785	0.739	0.999	0.984	0.548	1.040
GSS		0.000	1.427	0.983	0.001	0.814	0.000	0.213	0.325	0.780	0.099	0.006	0.401
GSS-R		0.000	1.404	0.948	0.001	0.832	0.000	0.202	0.332	0.790	0.082	0.006	0.417
NM		0.000	0.133	2.867	0.051	0.815	0.004	0.844	1.686	1.000	0.943	0.190	0.670
NM-R		0.000	0.131	1.140	0.460	0.505	0.000	0.326	0.782	0.999	0.646	0.000	0.161
PSO	–0.5/2	0.133	0.320	1.943	0.010	0.962	0.090	0.832	0.532	0.999	0.477	0.153	0.000
	1/2	2.735	0.320	4.892	64.875	0.984	8.484	1.590	1.615	1.000	0.549	2.819	0.000
REA-P	R1	0.029	0.786	0.918	0.496	0.966	0.691	0.644	0.445	0.999	0.987	0.091	0.165
	R2	0.057	0.600	1.418	0.650	0.966	0.803	0.747	0.310	0.999	0.856	0.040	0.255
REA-T	R1	0.005	0.544	1.552	0.004	0.942	0.009	0.608	0.221	0.949	0.909	0.051	0.027
	R2	0.029	0.486	1.314	0.005	0.950	0.023	0.604	0.272	0.705	0.961	0.069	0.000
SA		0.004	0.805	1.758	0.004	0.966	0.001	0.245	0.402	0.985	0.881	0.031	0.664
rBOA		0.000	0.049	0.164	0.001	0.959	0.002	0.076	0.040	0.974	1.407	0.000	1.212
rGA		0.000	0.407	0.989	0.002	0.848	0.004	0.194	0.115	0.999	0.187	0.023	0.639

Table A.26: Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 10 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.184	1.532	1.710	0.040	0.198	0.359	0.619	–	0.460	0.188	0.257
CG-R		0.000	0.099	0.605	0.001	0.312	0.457	0.112	0.086	0.066	0.149	0.013	0.095
CMA-ES	100	0.000	0.019	0.048	0.000	0.004	0.000	0.018	0.021	0.184	0.175	0.000	0.123
	750	0.000	0.002	0.021	0.000	0.002	0.000	0.001	0.000	0.216	0.135	0.000	0.219
	1250	0.000	0.029	0.071	0.000	0.017	0.000	0.067	0.000	0.126	0.081	0.001	0.084
	2500	0.085	0.047	0.217	0.025	0.001	0.069	0.089	0.074	0.003	0.089	0.061	0.069
CMA-ES-R	100	0.000	0.012	0.020	0.000	0.000	0.000	0.009	0.017	0.159	0.166	0.000	0.032
	750	0.000	0.021	0.021	0.000	0.008	0.000	0.004	0.000	0.202	0.150	0.002	0.192
	1250	0.001	0.030	0.065	0.000	0.003	0.000	0.072	0.000	0.183	0.076	0.002	0.080
	2500	0.082	0.047	0.222	0.023	0.001	0.067	0.087	0.061	0.000	0.082	0.067	0.070
DE	.2/.9	0.061	0.020	0.259	0.007	0.004	0.041	0.083	0.093	0.022	0.109	0.053	0.090
	.2/.2	0.001	0.019	0.119	0.000	0.026	0.006	0.068	0.063	0.017	0.117	0.002	0.088
	.9/.2	0.004	0.020	0.174	0.000	0.017	0.006	0.082	0.054	0.053	0.111	0.004	0.083
	.9/.9	0.133	0.021	0.280	0.083	0.001	0.142	0.093	0.086	0.003	0.078	0.140	0.065
GSS		0.000	0.186	0.281	0.000	0.147	0.000	0.084	0.165	0.174	0.137	0.004	0.240
GSS-R		0.000	0.191	0.308	0.000	0.090	0.000	0.078	0.176	0.172	0.127	0.004	0.251
NM		0.000	0.049	1.575	0.452	0.113	0.063	0.330	0.437	0.000	0.220	0.558	0.376
NM-R		0.000	0.046	0.695	3.420	0.320	0.000	0.125	0.201	0.001	0.144	0.000	0.070
PSO	–0.5/2	0.115	0.031	0.522	0.015	0.001	0.100	0.279	0.196	0.001	0.237	0.112	0.000
	1/2	0.826	0.036	0.683	69.350	0.002	4.853	0.200	0.222	–	0.014	0.818	0.000
REA-P	R1	0.009	0.057	0.194	0.586	0.001	0.444	0.082	0.064	0.003	0.074	0.017	0.030
	R2	0.066	0.050	0.503	0.777	0.001	0.493	0.104	0.053	0.001	0.103	0.011	0.191
REA-T	R1	0.005	0.044	0.194	0.000	0.011	0.004	0.081	0.047	0.042	0.086	0.013	0.062
	R2	0.013	0.043	0.203	0.000	0.007	0.008	0.087	0.042	0.132	0.082	0.017	0.000
SA		0.001	0.081	0.268	0.000	0.001	0.001	0.080	0.110	0.027	0.115	0.002	0.081
rBOA		0.000	0.062	0.051	0.000	0.002	0.000	0.092	0.000	0.031	0.129	0.000	0.080
rGA		0.000	0.093	0.398	0.000	0.060	0.006	0.072	0.017	0.010	0.104	0.002	0.043

Table A.27: Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 10 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	1.431	6.279	0.133	0.846	0.014	1.267	2.045	1.000	1.329	0.143	1.940
CG-R		0.000	0.817	2.645	0.002	0.249	0.000	0.359	0.009	0.994	0.552	0.010	1.237
CMA-ES	100	0.000	0.032	0.067	0.002	0.851	0.001	0.028	0.073	0.624	0.897	0.000	0.287
	750	0.000	0.000	0.010	0.002	0.851	0.002	0.000	0.040	0.401	0.482	0.000	0.087
	1250	0.000	0.000	0.002	0.001	0.854	0.002	0.000	0.040	0.367	0.397	0.000	0.045
	2500	0.000	0.000	0.001	0.001	0.854	0.002	0.000	0.040	0.264	0.983	0.000	1.005
CMA-ES-R	100	0.000	0.000	0.000	0.001	0.851	0.001	0.001	0.041	0.320	0.492	0.000	0.018
	750	0.000	0.071	0.017	0.001	0.872	0.002	0.005	0.040	0.211	0.362	0.015	0.264
	1250	0.001	0.120	0.041	0.002	0.961	0.003	0.012	0.040	0.299	0.373	0.017	0.353
	2500	0.039	0.158	0.078	0.003	0.960	0.016	0.046	0.040	0.993	0.998	0.037	1.001
DE	.2/.9	0.023	0.060	0.579	0.003	0.933	0.022	0.149	0.344	0.864	0.168	0.047	0.331
	.2/.2	0.000	0.061	0.035	0.000	0.846	0.002	0.072	0.117	0.872	0.155	0.003	0.225
	.9/.2	0.000	0.061	0.284	0.002	0.821	0.002	0.406	0.112	0.824	0.766	0.009	0.876
	.9/.9	0.369	0.062	2.301	0.044	0.964	0.215	0.677	0.629	0.997	0.888	0.367	0.958
GSS		0.000	1.427	0.983	0.001	0.814	0.000	0.213	0.325	0.780	0.099	0.006	0.401
GSS-R		0.000	0.094	0.724	0.000	0.780	0.000	0.145	0.187	0.610	0.016	0.002	0.208
NM		0.000	0.133	2.867	0.010	0.815	0.004	0.844	1.686	1.000	0.943	0.190	0.670
NM-R		0.000	0.114	0.553	0.001	0.237	0.000	0.200	0.589	0.998	0.507	0.000	0.083
PSO	–0.5/2	0.034	0.320	1.320	0.004	0.960	0.016	0.411	0.315	0.999	0.364	0.056	0.000
	1/2	2.668	0.320	4.885	64.875	0.984	8.484	1.590	1.615	1.000	0.548	2.741	0.000
REA-P	R1	0.008	0.682	0.178	0.495	0.965	0.691	0.479	0.263	0.997	0.880	0.046	0.098
	R2	0.001	0.439	0.310	0.650	0.965	0.803	0.438	0.152	0.997	0.543	0.024	0.189
REA-T	R1	0.000	0.321	0.672	0.002	0.762	0.001	0.410	0.040	0.472	0.689	0.015	0.027
	R2	0.000	0.273	0.450	0.003	0.744	0.001	0.412	0.041	0.345	0.815	0.016	0.000
SA		0.002	0.683	1.429	0.003	0.964	0.000	0.157	0.183	0.894	0.745	0.028	0.546
rBOA		0.000	0.001	0.079	0.001	0.955	0.002	0.002	0.040	0.917	1.407	0.000	1.117
rGA		0.000	0.403	0.988	0.000	0.842	0.002	0.186	0.093	0.991	0.185	0.021	0.602

Table A.28: Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 10 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.184	1.532	1.710	0.040	0.198	0.359	0.619	–	0.460	0.188	0.257
CG-R		0.000	0.085	0.435	0.000	0.336	0.000	0.087	0.021	0.066	0.107	0.005	0.082
CMA-ES	100	0.000	0.019	0.048	0.000	0.004	0.000	0.018	0.021	0.184	0.177	0.000	0.123
	750	0.000	0.002	0.021	0.000	0.001	0.000	0.001	–	0.217	0.151	0.000	0.087
	1250	0.000	0.000	0.010	0.000	0.009	0.000	0.000	0.000	0.203	0.132	0.000	0.062
	2500	0.000	0.000	0.006	0.000	0.009	0.000	0.000	0.000	0.180	0.085	0.000	0.096
CMA-ES-R	100	0.000	0.000	0.004	0.000	0.000	0.000	0.004	0.004	0.161	0.120	0.000	0.008
	750	0.000	0.016	0.005	0.000	0.004	0.000	0.001	0.000	0.169	0.108	0.001	0.032
	1250	0.000	0.024	0.013	0.000	0.006	0.000	0.004	0.000	0.192	0.139	0.002	0.051
	2500	0.060	0.036	0.025	0.000	0.002	0.019	0.025	0.000	0.010	0.074	0.025	0.106
DE	.2/.9	0.012	0.019	0.158	0.000	0.024	0.011	0.037	0.070	0.101	0.062	0.012	0.062
	.2/.2	0.000	0.017	0.038	0.000	0.074	0.002	0.028	0.027	0.097	0.086	0.001	0.066
	.9/.2	0.000	0.019	0.121	0.000	0.120	0.000	0.065	0.026	0.107	0.118	0.002	0.085
	.9/.9	0.099	0.017	0.241	0.029	0.001	0.084	0.088	0.081	0.003	0.075	0.098	0.059
GSS		0.000	0.186	0.281	0.000	0.147	0.000	0.084	0.165	0.174	0.137	0.004	0.240
GSS-R		0.000	0.022	0.190	0.000	0.175	0.000	0.046	0.064	0.139	0.035	0.002	0.191
NM		0.000	0.049	1.575	0.032	0.113	0.063	0.330	0.437	0.000	0.220	0.558	0.376
NM-R		0.000	0.034	0.290	0.000	0.323	0.000	0.069	0.129	0.010	0.103	0.000	0.052
PSO	–0.5/2	0.033	0.031	0.383	0.000	0.000	0.015	0.220	0.122	0.004	0.195	0.035	0.000
	1/2	0.885	0.036	0.694	69.350	0.003	4.853	0.200	0.222	–	0.015	0.897	0.000
REA-P	R1	0.002	0.047	0.083	0.586	0.002	0.445	0.075	0.039	0.007	0.067	0.005	0.010
	R2	0.000	0.033	0.188	0.777	0.001	0.493	0.177	0.022	0.008	0.103	0.001	0.133
REA-T	R1	0.000	0.035	0.100	0.000	0.247	0.000	0.061	0.000	0.019	0.084	0.001	0.062
	R2	0.000	0.030	0.106	0.000	0.260	0.000	0.066	0.004	0.214	0.073	0.002	0.000
SA		0.000	0.068	0.205	0.000	0.001	0.000	0.047	0.040	0.091	0.119	0.001	0.061
rBOA		0.000	0.009	0.031	0.000	0.004	0.000	0.015	0.003	0.052	0.129	0.000	0.078
rGA		0.000	0.093	0.398	0.000	0.060	0.000	0.072	0.015	0.059	0.104	0.002	0.035

Table A.29: Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 10 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	1.431	6.279	0.133	0.846	0.014	1.267	2.045	1.000	1.329	0.143	1.940
CG-R		0.000	0.752	2.294	0.002	0.060	0.000	0.303	0.001	0.989	0.469	0.006	1.164
CMA-ES	100	0.000	0.032	0.067	0.002	0.851	0.001	0.028	0.073	0.624	0.896	0.000	0.286
	750	0.000	0.000	0.010	0.002	0.851	0.002	0.000	0.040	0.401	0.482	0.000	0.087
	1250	0.000	0.000	0.002	0.001	0.854	0.002	0.000	0.040	0.367	0.394	0.000	0.040
	2500	0.000	0.000	0.000	0.001	0.851	0.001	0.000	0.040	0.264	0.210	0.000	0.012
CMA-ES-R	100	0.000	0.000	0.000	0.001	0.851	0.000	0.000	0.040	0.212	0.411	0.000	0.013
	750	0.000	0.059	0.013	0.001	0.868	0.002	0.004	0.040	0.087	0.239	0.013	0.238
	1250	0.001	0.106	0.035	0.001	0.957	0.003	0.009	0.040	0.147	0.242	0.016	0.299
	2500	0.008	0.144	0.067	0.002	0.958	0.007	0.022	0.040	0.982	0.324	0.026	0.389
DE	.2/.9	0.005	0.026	0.275	0.002	0.887	0.011	0.076	0.242	0.767	0.047	0.029	0.209
	.2/.2	0.000	0.029	0.034	0.000	0.825	0.002	0.021	0.075	0.740	0.079	0.001	0.047
	.9/.2	0.000	0.029	0.074	0.002	0.809	0.001	0.307	0.075	0.714	0.644	0.004	0.765
	.9/.9	0.282	0.029	2.130	0.021	0.962	0.151	0.620	0.573	0.993	0.821	0.296	0.910
GSS		0.000	1.427	0.983	0.001	0.814	0.000	0.213	0.325	0.780	0.099	0.006	0.401
GSS-R		0.000	0.077	0.594	0.000	0.701	0.000	0.113	0.136	0.515	0.000	0.001	0.120
NM		0.000	0.133	2.867	0.010	0.815	0.004	0.844	1.686	1.000	0.943	0.190	0.670
NM-R		0.000	0.102	0.337	0.000	0.040	0.000	0.154	0.495	0.997	0.432	0.000	0.047
PSO	–0.5/2	0.014	0.320	0.991	0.003	0.960	0.008	0.236	0.213	0.998	0.304	0.035	0.000
	1/2	2.646	0.320	4.875	64.875	0.984	8.484	1.590	1.615	1.000	0.547	2.694	0.000
REA-P	R1	0.004	0.616	0.114	0.495	0.964	0.691	0.367	0.190	0.994	0.822	0.036	0.075
	R2	0.000	0.364	0.304	0.650	0.964	0.803	0.235	0.108	0.996	0.276	0.021	0.176
REA-T	R1	0.000	0.216	0.094	0.001	0.762	0.000	0.081	0.040	0.163	0.326	0.004	0.027
	R2	0.000	0.127	0.059	0.002	0.736	0.000	0.082	0.040	0.243	0.683	0.010	0.000
SA		0.001	0.619	1.266	0.003	0.964	0.000	0.124	0.131	0.775	0.654	0.027	0.478
rBOA		0.000	0.000	0.059	0.001	0.952	0.002	0.000	0.040	0.864	1.407	0.000	1.064
rGA		0.000	0.401	0.988	0.000	0.841	0.002	0.181	0.082	0.984	0.183	0.020	0.574

Table A.30: Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 10 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.184	1.532	1.710	0.040	0.198	0.359	0.619	–	0.460	0.188	0.257
CG-R		0.000	0.082	0.388	0.000	0.193	0.000	0.069	0.007	0.084	0.089	0.003	0.081
CMA-ES	100	0.000	0.019	0.048	0.000	0.004	0.000	0.017	0.021	0.184	0.176	0.000	0.124
	750	0.000	0.002	0.021	0.000	0.000	0.000	0.001	–	0.217	0.151	0.000	0.087
	1250	0.000	0.000	0.010	0.000	0.009	0.000	0.000	–	0.203	0.131	0.000	0.061
	2500	0.000	0.000	0.006	0.000	0.000	0.000	0.000	0.000	0.181	0.085	0.000	0.032
CMA-ES-R	100	0.000	0.000	0.002	0.000	–	0.000	0.001	0.000	0.155	0.097	0.000	0.005
	750	0.000	0.013	0.003	0.000	0.003	0.000	0.001	0.000	0.099	0.078	0.001	0.025
	1250	0.000	0.020	0.009	0.000	0.013	0.000	0.002	–	0.133	0.084	0.001	0.031
	2500	0.011	0.022	0.018	0.000	0.003	0.003	0.007	0.000	0.019	0.131	0.004	0.050
DE	.2/.9	0.003	0.014	0.093	0.000	0.037	0.008	0.026	0.052	0.128	0.030	0.004	0.040
	.2/.2	0.000	0.015	0.038	0.000	0.127	0.001	0.015	0.022	0.137	0.073	0.000	0.046
	.9/.2	0.000	0.015	0.056	0.000	0.147	0.000	0.067	0.017	0.139	0.120	0.001	0.091
	.9/.9	0.084	0.015	0.218	0.012	0.002	0.059	0.076	0.071	0.013	0.075	0.080	0.058
GSS		0.000	0.186	0.281	0.000	0.147	0.000	0.084	0.165	0.174	0.137	0.004	0.240
GSS-R		0.000	0.014	0.149	0.000	0.252	0.000	0.034	0.035	0.084	0.007	0.001	0.169
NM		0.000	0.049	1.575	0.032	0.113	0.063	0.330	0.437	0.000	0.220	0.558	0.376
NM-R		0.000	0.026	0.174	0.000	0.160	0.000	0.048	0.098	0.013	0.090	0.000	0.036
PSO	–0.5/2	0.013	0.031	0.270	0.000	0.000	0.005	0.142	0.084	0.006	0.157	0.012	0.000
	1/2	0.909	0.036	0.701	69.350	0.003	4.853	0.200	0.222	–	0.014	0.966	0.000
REA-P	R1	0.001	0.048	0.060	0.587	0.001	0.445	0.062	0.027	0.012	0.066	0.003	0.007
	R2	0.000	0.028	0.189	0.777	0.001	0.493	0.090	0.017	0.008	0.089	0.001	0.127
REA-T	R1	0.000	0.025	0.084	0.000	0.249	0.000	0.033	0.000	0.202	0.074	0.003	0.062
	R2	0.000	0.049	0.041	0.000	0.279	0.000	0.042	0.000	0.220	0.074	0.002	0.000
SA		0.000	0.063	0.182	0.000	0.001	0.000	0.039	0.025	0.119	0.115	0.001	0.060
rBOA		0.000	0.000	0.029	0.000	0.004	0.000	0.001	0.004	0.072	0.129	0.000	0.070
rGA		0.000	0.093	0.399	0.000	0.061	0.000	0.071	0.006	0.081	0.104	0.002	0.035

Table A.31: Results on performance criterion ϕ_1 (scaled) in 10 dimensions.

ϕ_1 (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	1.431	6.280	0.348	0.847	0.062	1.267	2.045	1.000	1.331	0.160	1.940
CG-R		0.000	0.827	2.719	0.003	0.263	0.090	0.372	0.036	0.994	0.573	0.027	1.241
CMA-ES	100	0.000	0.033	0.069	0.002	0.852	0.001	0.031	0.073	0.625	0.902	0.000	0.298
	750	0.009	0.023	0.073	0.004	0.856	0.009	0.030	0.055	0.429	0.574	0.008	0.216
	1250	0.023	0.049	0.140	0.006	0.864	0.019	0.060	0.073	0.425	0.584	0.020	0.294
	2500	0.061	0.133	0.375	0.008	0.881	0.037	0.151	0.130	0.425	0.665	0.054	0.579
CMA-ES-R	100	0.000	0.003	0.006	0.001	0.852	0.001	0.005	0.044	0.316	0.511	0.000	0.034
	750	0.009	0.089	0.078	0.003	0.875	0.009	0.033	0.054	0.232	0.426	0.022	0.344
	1250	0.022	0.153	0.170	0.005	0.961	0.017	0.068	0.072	0.330	0.483	0.034	0.474
	2500	0.087	0.246	0.433	0.009	0.961	0.047	0.170	0.130	0.991	0.718	0.084	0.733
DE	.2/.9	0.053	0.061	0.674	0.015	0.925	0.046	0.182	0.358	0.859	0.198	0.077	0.358
	.2/.2	0.014	0.062	0.168	0.009	0.855	0.016	0.107	0.151	0.853	0.201	0.020	0.250
	.9/.2	0.011	0.062	0.391	0.005	0.836	0.010	0.414	0.142	0.818	0.769	0.022	0.878
	.9/.9	0.386	0.063	2.340	0.067	0.964	0.242	0.687	0.639	0.996	0.890	0.394	0.963
GSS		0.000	1.427	0.985	0.001	0.815	0.000	0.213	0.325	0.780	0.099	0.007	0.402
GSS-R		0.000	0.250	0.703	0.000	0.763	0.000	0.142	0.187	0.598	0.021	0.003	0.209
NM		0.000	0.133	2.867	0.048	0.815	0.009	0.844	1.686	1.000	0.943	0.191	0.670
NM-R		0.000	0.114	0.656	0.222	0.238	0.002	0.220	0.615	0.998	0.518	0.002	0.094
PSO	–0.5/2	0.055	0.320	1.366	0.009	0.960	0.035	0.441	0.332	0.999	0.376	0.075	0.000
	1/2	2.681	0.320	4.886	64.875	0.984	8.484	1.590	1.615	1.000	0.548	2.737	0.000
REA-P	R1	0.022	0.688	0.362	0.496	0.965	0.691	0.486	0.287	0.996	0.888	0.062	0.127
	R2	0.033	0.456	0.547	0.650	0.965	0.803	0.417	0.182	0.997	0.540	0.049	0.227
REA-T	R1	0.012	0.342	0.740	0.006	0.800	0.011	0.351	0.091	0.518	0.642	0.034	0.037
	R2	0.020	0.290	0.492	0.007	0.790	0.015	0.376	0.101	0.386	0.812	0.040	0.022
SA		0.003	0.691	1.465	0.004	0.965	0.000	0.177	0.226	0.879	0.753	0.029	0.554
rBOA		0.000	0.015	0.101	0.002	0.955	0.002	0.027	0.041	0.914	1.407	0.000	1.121
rGA		0.004	0.406	0.999	0.004	0.848	0.006	0.189	0.104	0.991	0.187	0.024	0.605

Table A.32: Variance for performance criterion ϕ_1 (scaled) in 10 dimensions.

$Var(\phi_1)$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.184	1.533	1.700	0.040	0.195	0.359	0.619	–	0.460	0.183	0.257
CG-R		0.000	0.067	0.348	0.000	0.214	0.060	0.066	0.025	0.060	0.081	0.004	0.063
CMA-ES	100	0.000	0.019	0.047	0.000	0.004	0.000	0.018	0.021	0.184	0.174	0.000	0.122
	750	0.001	0.002	0.021	0.001	0.001	0.002	0.002	0.001	0.206	0.136	0.001	0.081
	1250	0.004	0.001	0.014	0.003	0.009	0.005	0.004	0.002	0.183	0.106	0.002	0.062
	2500	0.010	0.006	0.025	0.004	0.003	0.010	0.012	0.009	0.137	0.064	0.008	0.064
CMA-ES-R	100	0.000	0.001	0.004	0.000	0.000	0.000	0.002	0.003	0.131	0.091	0.000	0.008
	750	0.001	0.012	0.005	0.001	0.003	0.001	0.002	0.001	0.118	0.075	0.002	0.026
	1250	0.004	0.019	0.013	0.002	0.005	0.004	0.005	0.002	0.134	0.075	0.003	0.037
	2500	0.027	0.021	0.029	0.003	0.001	0.011	0.016	0.007	0.009	0.079	0.014	0.059
DE	.2/.9	0.013	0.014	0.111	0.008	0.021	0.013	0.029	0.047	0.080	0.041	0.012	0.046
	.2/.2	0.002	0.013	0.036	0.006	0.074	0.005	0.022	0.020	0.078	0.071	0.003	0.045
	.9/.2	0.002	0.014	0.075	0.002	0.102	0.003	0.049	0.017	0.090	0.097	0.003	0.066
	.9/.9	0.073	0.013	0.186	0.026	0.001	0.065	0.065	0.058	0.008	0.057	0.075	0.044
GSS		0.000	0.186	0.280	0.000	0.146	0.000	0.084	0.165	0.173	0.137	0.004	0.240
GSS-R		0.000	0.028	0.148	0.000	0.159	0.000	0.035	0.046	0.097	0.025	0.001	0.164
NM		0.001	0.049	1.575	0.227	0.112	0.065	0.330	0.437	–	0.220	0.559	0.376
NM-R		0.000	0.031	0.235	1.930	0.207	0.023	0.053	0.099	0.009	0.080	0.008	0.037
PSO	–0.5/2	0.032	0.031	0.270	0.013	0.000	0.021	0.150	0.085	0.004	0.168	0.030	0.000
	1/2	0.860	0.036	0.690	69.350	0.003	4.853	0.200	0.222	–	0.014	0.891	0.000
REA-P	R1	0.003	0.031	0.064	0.586	0.001	0.444	0.048	0.022	0.007	0.050	0.004	0.008
	R2	0.008	0.020	0.168	0.777	0.001	0.493	0.087	0.013	0.006	0.058	0.006	0.131
REA-T	R1	0.002	0.015	0.050	0.001	0.188	0.002	0.041	0.006	0.060	0.044	0.003	0.062
	R2	0.003	0.020	0.051	0.002	0.195	0.002	0.043	0.006	0.151	0.045	0.003	0.002
SA		0.000	0.051	0.164	0.000	0.001	0.000	0.039	0.034	0.072	0.097	0.001	0.048
rBOA		0.000	0.011	0.028	0.000	0.003	0.000	0.018	0.002	0.044	0.129	0.000	0.061
rGA		0.001	0.093	0.397	0.003	0.057	0.002	0.071	0.009	0.051	0.103	0.002	0.029

Table A.33: Results on performance criterion ϕ_2 (scaled) in 10 dimensions.

ϕ_2 (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	1.431	6.729	24.032	0.914	2.809	1.267	2.059	1.000	1.859	1.373	1.940
CG-R		0.000	1.276	6.006	0.071	0.901	4.561	0.986	1.330	1.000	1.397	1.030	1.712
CMA-ES	100	0.013	0.371	0.892	0.003	0.960	0.011	0.609	0.202	0.990	1.418	0.033	1.476
	750	0.702	0.858	2.765	0.261	0.975	0.601	0.939	0.798	1.000	1.384	0.623	1.441
	1250	0.692	0.876	2.863	0.276	0.976	0.599	0.945	0.813	1.000	1.388	0.673	1.424
	2500	0.541	0.826	2.634	0.120	0.972	0.379	0.838	0.721	1.000	1.302	0.532	1.250
CMA-ES-R	100	0.010	0.366	0.863	0.003	0.957	0.009	0.590	0.193	0.956	1.380	0.031	1.310
	750	0.628	0.838	2.700	0.191	0.972	0.509	0.894	0.760	1.000	1.338	0.583	1.278
	1250	0.621	0.847	2.734	0.166	0.972	0.475	0.875	0.776	1.000	1.345	0.587	1.278
	2500	0.552	0.819	2.632	0.117	0.972	0.398	0.826	0.729	1.000	1.314	0.537	1.242
DE	.2/.9	1.190	0.227	3.491	1.937	0.972	1.451	1.056	1.088	0.999	1.129	1.152	1.140
	.2/.2	1.135	0.228	3.490	1.988	0.971	1.498	1.050	1.083	1.000	1.120	1.163	1.154
	.9/.2	0.800	0.228	3.179	0.708	0.969	0.934	1.010	0.963	0.999	1.212	0.866	1.293
	.9/.9	1.116	0.225	3.561	1.577	0.971	1.417	1.085	1.031	0.999	1.216	1.097	1.241
GSS		0.135	1.447	1.479	0.007	0.952	0.074	0.309	0.357	0.836	0.369	0.160	0.488
GSS-R		0.145	1.432	1.430	0.007	0.953	0.079	0.300	0.361	0.849	0.364	0.161	0.507
NM		0.086	0.133	2.873	7.862	0.832	0.850	0.846	1.686	1.000	0.973	0.242	0.670
NM-R		0.074	0.140	3.277	5.600	0.810	0.457	0.829	1.478	0.999	0.963	0.229	0.455
PSO	–0.5/2	0.388	0.320	2.633	0.168	0.965	0.342	1.088	0.811	1.000	0.844	0.397	0.000
	1/2	2.821	0.320	4.910	64.878	0.984	8.484	1.591	1.615	1.000	0.638	2.885	0.000
REA-P	R1	0.849	1.014	3.300	0.611	0.972	0.780	1.056	0.974	1.000	1.220	0.767	1.157
	R2	1.005	0.978	3.540	0.758	0.971	0.907	1.087	0.958	1.000	1.189	1.146	1.207
REA-T	R1	0.744	0.967	3.296	0.638	0.969	0.717	1.034	0.941	1.000	1.223	0.852	1.033
	R2	0.859	0.947	3.224	0.785	0.969	0.865	1.038	0.940	0.999	1.218	0.918	1.130
SA		0.038	1.067	2.801	0.111	0.972	0.007	1.040	1.225	1.000	1.173	0.041	0.975
rBOA		0.186	0.376	0.926	0.055	0.968	0.159	0.831	0.196	1.000	1.407	0.011	1.309
rGA		0.582	0.828	2.533	1.105	0.966	0.797	0.675	0.846	1.000	0.698	0.634	0.954

Table A.34: Variance for performance criterion ϕ_2 (scaled) in 10 dimensions.

$Var(\phi_2)$ (scaled)		sphr.	ack.	lg-ack.	whit.	shek.	rosen.	rastr.	sal.	lange.	schw.	grie.	weier.
CG		0.000	0.184	2.104	1.922	0.031	–	0.359	0.638	–	0.739	0.000	0.257
CG-R		0.000	0.138	1.872	0.004	0.036	0.359	0.293	0.701	0.000	0.355	0.000	0.186
CMA-ES	100	0.004	0.032	0.105	0.000	0.001	0.003	0.078	0.032	0.010	0.092	0.002	0.097
	750	0.169	0.065	0.320	0.198	0.002	0.251	0.116	0.101	0.000	0.089	0.150	0.092
	1250	0.162	0.054	0.301	0.219	0.002	0.242	0.089	0.092	0.000	0.085	0.140	0.087
	2500	0.137	0.059	0.281	0.089	0.002	0.160	0.097	0.098	0.000	0.091	0.133	0.099
CMA-ES-R	100	0.002	0.030	0.105	0.000	0.002	0.002	0.092	0.027	0.038	0.099	0.002	0.090
	750	0.154	0.063	0.285	0.134	0.001	0.219	0.102	0.096	0.000	0.094	0.138	0.092
	1250	0.157	0.062	0.246	0.126	0.002	0.178	0.104	0.099	0.000	0.088	0.129	0.087
	2500	0.124	0.061	0.269	0.083	0.002	0.142	0.116	0.091	0.000	0.078	0.129	0.091
DE	.2/.9	0.311	0.033	0.419	1.571	0.002	0.657	0.143	0.135	0.000	0.121	0.294	0.099
	.2/.2	0.294	0.029	0.426	1.583	0.002	0.696	0.136	0.132	0.000	0.124	0.308	0.097
	.9/.2	0.269	0.035	0.380	0.709	0.002	0.535	0.130	0.137	0.000	0.112	0.245	0.081
	.9/.9	0.262	0.032	0.355	1.254	0.002	0.595	0.123	0.119	0.001	0.100	0.260	0.089
GSS		0.061	0.184	0.266	0.005	0.011	0.056	0.094	0.151	0.133	0.150	0.064	0.218
GSS-R		0.059	0.179	0.286	0.003	0.010	0.050	0.089	0.165	0.123	0.143	0.062	0.227
NM		0.377	0.049	1.580	39.019	0.101	3.355	0.332	0.437	0.000	0.217	0.671	0.376
NM-R		0.327	0.053	1.482	31.496	0.126	2.533	0.332	0.370	0.000	0.210	0.627	0.201
PSO	–0.5/2	0.256	0.031	0.524	0.429	0.003	0.326	0.158	0.183	0.000	0.180	0.214	0.000
	1/2	0.744	0.036	0.668	69.348	0.002	4.853	0.200	0.221	–	0.134	0.787	0.000
REA-P	R1	0.254	0.074	0.319	0.620	0.002	0.420	0.115	0.129	0.000	0.096	0.195	0.094
	R2	0.261	0.084	0.396	0.841	0.002	0.482	0.127	0.128	0.000	0.101	0.276	0.084
REA-T	R1	0.180	0.078	0.324	0.529	0.002	0.304	0.111	0.100	0.000	0.099	0.188	0.088
	R2	0.208	0.066	0.320	0.576	0.001	0.336	0.121	0.119	0.000	0.096	0.194	0.081
SA		0.017	0.090	0.419	0.301	0.002	0.010	0.288	0.190	0.000	0.107	0.005	0.109
rBOA		0.175	0.068	0.193	0.129	0.002	0.184	0.207	0.049	0.000	0.129	0.003	0.101
rGA		0.216	0.089	0.400	1.266	0.002	0.437	0.110	0.136	0.000	0.121	0.203	0.072

Table A.35: Results on performance criterion σ_ϵ^N with $\epsilon = 10.000$ and $N = 250,000$ in 25 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		1.000	1.000	0.000	0.000	0.970	0.000	0.965	0.000	1.000	0.000
CG-R		1.000	1.000	0.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
CMA-ES	100	1.000	1.000	0.995	0.000	1.000	0.835	1.000	0.000	1.000	0.005
	750	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.004
	1250	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.010
	2500	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.000
CMA-ES-R	100	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.080
	750	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.000
	1250	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.005
	2500	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
DE	.2/.9	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
	.2/.2	1.000	1.000	0.705	0.005	0.005	0.030	1.000	0.000	0.680	0.000
	.9/.2	1.000	1.000	0.075	0.000	0.010	0.000	1.000	0.000	1.000	0.000
	.9/.9	0.040	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
GSS		1.000	1.000	0.000	0.005	1.000	0.000	1.000	0.950	1.000	0.985
GSS-R		1.000	1.000	0.000	0.000	1.000	0.000	1.000	0.990	1.000	0.990
NM		0.985	1.000	0.000	0.000	0.465	0.000	1.000	0.000	0.400	0.000
NM-R		0.980	1.000	0.000	0.000	0.565	0.000	1.000	0.000	0.455	0.010
PSO	–0.5/2	1.000	1.000	0.000	0.000	0.000	0.059	1.000	0.000	1.000	1.000
	1/2	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	1.000
REA-P	R1	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
	R2	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
REA-T	R1	1.000	1.000	0.255	0.000	0.000	0.000	1.000	0.000	1.000	0.545
	R2	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000
	R3	0.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.005
SA		1.000	1.000	0.000	0.000	0.045	0.000	1.000	0.000	1.000	0.000
rBOA		0.999	1.000	0.000	0.000	0.000	0.000	1.000	0.000	0.999	0.000
rGA		1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000

Table A.36: Results on performance criterion σ_ϵ^N with $\epsilon = 1.000$ and $N = 250,000$ in 25 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.880	0.000	0.010	0.000	1.000	0.000
CG-R		1.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000
CMA-ES	100	1.000	1.000	0.005	0.000	0.020	0.000	1.000	0.000	1.000	0.000
	750	1.000	1.000	0.690	0.000	0.000	0.970	1.000	0.000	1.000	0.000
	1250	1.000	1.000	0.905	0.000	0.000	1.000	1.000	0.000	1.000	0.000
	2500	1.000	1.000	1.000	0.000	0.000	1.000	1.000	0.000	1.000	0.000
CMA-ES-R	100	1.000	1.000	0.305	0.000	0.000	0.035	1.000	0.000	1.000	0.000
	750	1.000	1.000	0.755	0.000	0.000	0.980	1.000	0.000	1.000	0.000
	1250	0.059	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
	2500	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
DE	.2/.9	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	.2/.2	1.000	1.000	0.010	0.000	0.000	0.000	0.860	0.000	0.380	0.000
	.9/.2	1.000	1.000	0.000	0.000	0.000	0.000	0.975	0.000	0.965	0.000
	.9/.9	0.000	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.010	0.000	0.005	0.665	0.000	0.225	0.770	1.000	0.610
GSS-R		1.000	0.985	0.000	0.000	0.625	0.000	0.504	0.960	1.000	0.665
NM		0.965	1.000	0.000	0.000	0.215	0.000	0.000	0.000	0.140	0.000
NM-R		0.975	1.000	0.000	0.000	0.290	0.000	0.000	0.000	0.142	0.000
PSO	–0.5/2	0.870	0.000	0.000	0.000	0.000	0.000	0.755	0.000	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
REA-T	R1	1.000	1.000	0.000	0.000	0.000	0.000	1.000	0.000	1.000	0.000
	R2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
SA		1.000	0.000	0.000	0.000	0.000	0.000	0.120	0.000	0.000	0.000
rBOA		0.999	0.140	0.000	0.000	0.000	0.000	1.000	0.000	0.995	0.000
rGA		1.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000

Table A.37: Results on performance criterion σ_ϵ^N with $\epsilon = 0.100$ and $N = 250,000$ in 25 dimensions.

σ_ϵ^N		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		1.000	0.000	0.000	0.000	0.880	0.000	0.005	0.000	0.990	0.000
CG-R		1.000	0.000	0.000	0.000	1.000	0.000	0.930	0.000	1.000	0.000
CMA-ES	100	1.000	0.045	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	750	1.000	1.000	0.320	0.000	0.000	0.750	0.040	0.000	1.000	0.000
	1250	1.000	1.000	0.545	0.000	0.000	0.985	0.205	0.000	1.000	0.000
	2500	1.000	1.000	0.820	0.000	0.000	1.000	0.725	0.000	1.000	0.000
CMA-ES-R	100	1.000	0.640	0.045	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	750	1.000	1.000	0.030	0.000	0.000	0.335	0.020	0.000	0.000	0.000
	1250	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	2500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
DE	.2/.9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	.2/.2	0.990	0.000	0.005	0.000	0.000	0.000	0.000	0.000	0.025	0.000
	.9/.2	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.175	0.000
	.9/.9	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
GSS		1.000	0.000	0.000	0.005	0.615	0.000	0.000	0.770	1.000	0.345
GSS-R		1.000	0.000	0.000	0.000	0.570	0.000	0.000	0.960	1.000	0.430
NM		0.955	0.000	0.000	0.000	0.155	0.000	0.000	0.000	0.005	0.000
NM-R		0.970	0.000	0.000	0.000	0.165	0.000	0.000	0.000	0.010	0.000
PSO	–0.5/2	0.105	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
	1/2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000
REA-P	R1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
REA-T	R1	1.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
	R2	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	R3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
SA		0.010	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
rBOA		0.999	0.000	0.000	0.000	0.000	0.000	0.049	0.000	0.746	0.000
rGA		0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

Table A.38: Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 10.000$ and $N = 250,000$ in 25 dimensions.

$\frac{1}{100}\hat{\psi}_\epsilon^N$		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.0	0.0	–	–	216.1	–	4.2	–	0.0	–
CG-R		0.0	0.0	–	–	226.3	–	4.9	–	0.0	–
CMA-ES	100	10.2	0.0	45.0	–	399.3	55.6	0.0	–	14.4	173.0
	750	82.0	0.0	244.3	–	–	262.7	0.0	–	101.4	735.0
	1250	144.5	0.0	387.1	–	–	411.7	0.0	–	171.7	1400.0
	2500	303.2	0.0	750.3	–	–	787.1	0.0	–	362.8	–
CMA-ES-R	100	9.5	0.0	45.1	–	–	72.7	0.0	–	13.7	981.6
	750	32.6	0.0	119.7	–	–	129.0	0.0	–	46.3	–
	1250	134.3	0.0	–	–	–	–	0.0	–	29.3	2104.0
	2500	303.3	0.0	–	–	–	–	0.0	–	14.3	–
DE	.2/.9	1097.7	0.0	–	–	–	–	0.0	–	–	–
	.2/.2	146.5	0.0	1066.1	1229.0	1778.0	2222.1	0.0	–	1842.4	–
	.9/.2	128.0	0.0	2279.5	–	2234.0	–	0.0	–	911.0	–
	.9/.9	2017.6	0.0	–	–	–	–	0.0	–	–	–
GSS		66.7	0.0	–	210.0	650.0	–	0.0	126.8	105.1	36.2
GSS-R		65.8	0.0	–	–	622.4	–	0.0	197.3	104.1	33.1
NM		116.6	0.0	–	–	488.1	–	0.0	–	20.5	25.0
NM-R		105.4	0.0	–	–	617.8	–	0.0	–	20.1	24.8
PSO	–0.5/2	179.6	0.0	–	–	–	1521.8	0.0	–	534.2	1.3
	1/2	–	0.0	–	–	–	–	0.0	–	–	0.0
REA-P	R1	–	0.0	–	–	–	–	0.0	–	–	–
	R2	–	0.0	–	–	–	–	0.0	–	–	–
REA-T	R1	27.0	0.0	799.1	–	–	–	0.0	–	74.3	121.5
	R2	–	0.0	–	–	–	–	0.0	–	–	–
	R3	–	0.0	–	–	–	–	0.0	–	–	2434.0
SA		2.9	0.0	–	–	1443.7	–	0.0	–	5.0	–
rBOA		91.6	0.0	–	–	–	–	0.0	–	402.0	–
rGA		61.2	0.0	–	–	–	–	0.0	–	76.5	–

Table A.39: Results on performance criterion $\frac{1}{100}\hat{\psi}_\epsilon^N$ with $\epsilon = 1.000$ and $N = 250,000$ in 25 dimensions.

$\frac{1}{100}\hat{\psi}_\epsilon^N$		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.0	–	–	–	217.0	–	4.5	–	0.0	–
CG-R		0.0	–	–	–	239.7	–	194.0	–	0.0	–
CMA-ES	100	18.4	39.5	62.0	–	2210.0	–	19.8	–	33.0	–
	750	125.3	213.1	344.8	–	–	321.4	124.5	–	203.9	–
	1250	207.9	353.7	525.1	–	–	475.1	208.2	–	330.7	–
	2500	406.2	710.1	943.5	–	–	889.6	421.3	–	626.0	–
CMA-ES-R	100	17.6	38.7	1189.7	–	–	1216.0	18.9	–	32.2	–
	750	53.8	99.0	1069.3	–	–	689.6	54.7	–	96.3	–
	1250	299.5	–	–	–	–	–	201.0	–	61.9	–
	2500	–	–	–	–	–	–	422.0	–	33.1	–
DE	.2/.9	–	181.3	–	–	–	–	–	–	–	–
	.2/.2	361.2	179.2	1942.5	–	–	–	1375.5	–	2168.9	–
	.9/.2	466.1	190.3	–	–	–	–	1334.5	–	1823.0	–
	.9/.9	–	178.6	–	–	–	–	–	–	–	–
GSS		131.9	68.5	–	270.0	1193.9	–	51.7	184.9	281.5	294.0
GSS-R		131.5	1497.6	–	–	1213.7	–	888.6	429.0	281.9	261.3
NM		139.5	4.1	–	–	547.0	–	–	–	24.3	25.0
NM-R		139.5	4.1	–	–	801.6	–	–	–	24.3	25.0
PSO	–0.5/2	1005.0	–	–	–	–	–	1013.9	–	–	5.6
	1/2	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	–	–	–	–	–	–	–	–	–	–
	R2	–	–	–	–	–	–	–	–	–	–
REA-T	R1	85.2	825.1	–	–	–	–	90.5	–	499.4	–
	R2	–	–	–	–	–	–	–	–	–	–
	R3	–	–	–	–	–	–	–	–	–	–
SA		80.0	–	–	–	–	–	2019.1	–	–	–
rBOA		303.9	1345.8	–	–	–	–	38.7	–	1515.2	–
rGA		118.2	–	–	–	–	–	284.4	–	–	–

Table A.40: Results on performance criterion $\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$ with $\epsilon = 0.100$ and $N = 250,000$ in 25 dimensions.

$\frac{1}{100}\hat{\psi}_{\epsilon}^{N}$		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.0	–	–	–	217.0	–	4.0	–	238.9	–
CG-R		0.0	–	–	–	239.7	–	827.4	–	240.2	–
CMA-ES	100	25.4	52.8	–	–	–	–	–	–	43.0	–
	750	163.8	268.1	319.2	–	–	335.5	244.3	–	262.7	–
	1250	266.3	434.8	517.8	–	–	530.9	421.1	–	425.0	–
	2500	503.5	849.7	991.9	–	–	988.0	887.5	–	794.5	–
CMA-ES-R	100	24.5	1030.9	1667.8	–	–	–	–	–	42.2	–
	750	72.8	177.1	1368.5	–	–	1256.6	1167.7	–	–	–
	1250	–	–	–	–	–	–	–	–	78.7	–
	2500	–	–	–	–	–	–	–	–	43.1	–
DE	.2/.9	–	–	–	–	–	–	–	–	–	–
	.2/.2	643.0	–	1796.0	–	–	–	–	–	2366.8	–
	.9/.2	823.9	–	–	–	–	–	–	–	2390.4	–
	.9/.9	–	–	–	–	–	–	–	–	–	–
GSS		197.4	–	–	323.0	1534.1	–	–	250.0	336.3	459.0
GSS-R		197.5	–	–	–	1544.2	–	–	494.2	335.4	412.2
NM		149.7	–	–	–	713.9	–	–	–	24.9	25.0
NM-R		168.7	–	–	–	1134.9	–	–	–	24.9	25.0
PSO	–0.5/2	1478.7	–	–	–	–	–	–	–	–	7.1
	1/2	–	–	–	–	–	–	–	–	–	0.0
REA-P	R1	–	–	–	–	–	–	–	–	–	–
	R2	–	–	–	–	–	–	–	–	–	–
REA-T	R1	183.4	–	–	–	–	–	–	–	965.3	–
	R2	–	–	–	–	–	–	–	–	–	–
	R3	–	–	–	–	–	–	–	–	–	–
SA		1666.5	–	–	–	–	–	–	–	–	–
rBOA		579.2	–	–	–	–	–	2029.2	–	1849.1	–
rGA		349.0	–	–	–	–	–	–	–	–	–

Table A.41: Results on performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 25 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	1.050	2.313	0.009	0.139	0.793	1.428	1.141	0.000	1.454
CG-R		0.000	0.872	1.818	0.000	0.000	0.531	0.175	0.892	0.000	1.204
CMA-ES	100	0.000	0.032	0.030	0.000	0.000	0.029	0.058	0.805	0.000	∞
	750	0.000	0.041	0.043	0.000	0.000	0.064	0.029	1.134	0.000	∞
	1250	0.003	0.287	0.330	0.000	0.001	0.493	0.111	1.127	0.005	∞
	2500	0.455	0.713	1.141	0.060	0.246	0.760	0.599	1.063	0.365	∞
CMA-ES-R	100	0.000	0.027	0.023	0.000	0.000	0.022	0.061	0.739	0.000	∞
	750	0.000	0.012	0.015	0.000	0.000	0.006	0.036	0.946	0.001	∞
	1250	0.019	0.304	0.327	0.000	0.002	0.488	0.103	1.084	0.000	∞
	2500	0.447	0.717	1.147	0.060	0.243	0.767	0.600	1.058	0.000	∞
DE	.2/.9	0.278	0.167	1.002	0.021	0.123	0.545	0.657	0.657	1.683	1.263
	.2/.2	0.038	0.167	0.497	0.000	0.014	0.438	0.432	0.680	0.439	0.905
	.9/.2	0.049	0.168	0.713	0.000	0.013	0.644	0.373	0.987	0.352	1.186
	.9/.9	0.441	0.168	1.280	0.073	0.240	0.818	0.716	0.982	∞	∞
GSS		0.000	0.869	0.554	0.000	0.000	0.277	0.253	0.013	0.003	0.047
GSS-R		0.000	0.861	0.537	0.000	0.000	0.282	0.285	0.017	0.003	0.041
NM		0.026	0.088	1.266	0.178	0.094	0.636	1.331	0.709	–	–
NM-R		0.025	0.087	1.244	0.198	0.061	0.637	1.084	0.698	–	–
PSO	–0.5/2	0.075	0.300	0.735	0.000	0.015	0.560	0.341	0.680	0.071	0.000
	1/2	1.782	0.297	2.264	13.294	3.635	1.319	1.363	0.416	1.795	0.000
REA-P	R1	0.708	0.898	1.533	0.244	0.448	0.864	0.821	0.997	0.683	1.046
	R2	0.627	0.877	1.558	0.231	0.449	0.892	0.821	0.992	0.707	1.060
REA-T	R1	0.000	0.313	0.323	0.000	0.000	0.256	0.065	0.729	0.004	0.287
	R2	0.639	0.873	1.476	0.224	0.441	0.874	0.810	0.987	0.654	1.010
	R3	0.618	0.865	1.462	0.224	0.419	0.857	0.800	0.987	0.638	0.983
SA		0.005	0.873	0.979	0.000	0.000	0.447	0.638	0.958	0.011	0.759
rBOA		0.021	0.282	0.326	0.000	0.004	0.568	0.088	1.178	0.078	1.169
rGA		0.002	0.486	0.578	0.000	0.001	0.309	0.173	0.308	0.010	0.776

Table A.42: Variance for performance criterion ζ_{T_m} (scaled) with $m = 25,000$ in 25 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	0.074	0.356	0.050	0.383	0.139	0.277	0.285	0.000	0.112
CG-R		0.000	0.059	0.219	0.000	0.000	0.067	0.215	0.203	0.000	0.056
CMA-ES	100	0.000	0.008	0.010	0.000	0.000	0.009	0.010	0.097	0.000	∞
	750	0.000	0.009	0.016	0.000	0.000	0.025	0.007	0.034	0.000	∞
	1250	0.000	0.013	0.022	0.000	0.000	0.035	0.009	0.034	0.000	∞
	2500	0.056	0.030	0.060	0.026	0.058	0.045	0.039	0.035	0.049	∞
CMA-ES-R	100	0.000	0.004	0.112	0.017	0.006	0.148	0.022	150.146	0.000	∞
	750	0.000	0.001	0.055	0.015	0.008	0.018	0.007	245.633	0.000	∞
	1250	0.034	0.513	19.894	0.072	0.520	65.416	0.061	318.209	0.000	∞
	2500	17.242	2.840	244.431	21978.049	3971.182	161.246	2.084	303.517	0.000	∞
DE	.2/.9	0.056	0.018	0.099	0.014	0.038	0.059	0.063	0.062	0.237	0.059
	.2/.2	0.010	0.017	0.063	0.000	0.005	0.055	0.060	0.068	0.085	0.060
	.9/.2	0.013	0.017	0.072	0.000	0.005	0.050	0.046	0.046	0.083	0.044
	.9/.9	0.089	0.017	0.110	0.049	0.084	0.054	0.064	0.049	∞	∞
GSS		0.000	0.155	0.115	0.000	0.000	0.089	0.090	0.061	0.000	0.067
GSS-R		0.000	0.151	0.102	0.000	0.000	0.111	0.136	0.087	0.000	0.065
NM		0.077	0.024	0.404	0.699	0.278	0.146	0.145	0.104	–	–
NM-R		0.074	0.025	0.402	0.691	0.199	0.153	0.090	0.097	–	–
PSO	–0.5/2	0.070	0.014	0.201	0.001	0.023	0.224	0.131	0.149	0.062	0.000
	1/2	0.324	0.014	0.175	4.117	0.738	0.080	0.088	0.029	0.258	0.000
REA-P	R1	0.074	0.032	0.076	0.117	0.107	0.047	0.043	0.037	0.078	0.035
	R2	0.069	0.030	0.074	0.099	0.102	0.044	0.048	0.038	0.076	0.035
REA-T	R1	0.000	0.016	0.034	0.000	0.000	0.040	0.010	0.037	0.000	0.062
	R2	0.064	0.030	0.076	0.084	0.092	0.043	0.046	0.039	0.075	0.035
	R3	0.069	0.035	0.076	0.084	0.089	0.044	0.049	0.036	0.068	0.035
SA		0.001	0.051	0.095	0.000	0.000	0.098	0.096	0.047	0.001	0.065
rBOA		0.017	0.030	0.033	0.000	0.003	0.069	0.016	0.063	0.042	0.048
rGA		0.000	0.057	0.124	0.000	0.000	0.041	0.020	0.064	0.000	0.029

Table A.43: Results on performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 25 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	1.050	2.313	0.001	0.030	0.793	1.428	1.141	0.000	1.454
CG-R		0.000	0.826	1.593	0.000	0.000	0.455	0.028	0.658	0.000	1.146
CMA-ES	100	0.000	0.032	0.030	0.000	0.000	0.029	0.058	0.805	0.000	∞
	750	0.000	0.000	0.005	0.000	0.000	0.001	0.028	0.583	0.000	∞
	1250	0.000	0.000	0.003	0.000	0.000	0.000	0.020	1.024	0.000	∞
	2500	0.000	0.001	0.001	0.000	0.000	0.000	0.017	1.054	0.000	∞
CMA-ES-R	100	0.000	0.018	0.013	0.000	0.000	0.014	0.048	0.595	0.000	∞
	750	0.000	0.003	0.005	0.000	0.000	0.001	0.033	0.547	0.000	∞
	1250	0.018	0.296	0.275	0.000	0.002	0.149	0.047	0.913	0.000	∞
	2500	0.036	0.362	0.416	0.000	0.006	0.361	0.120	1.052	0.000	∞
DE	.2/.9	0.122	0.126	0.702	0.001	0.035	0.359	0.498	0.449	1.649	1.252
	.2/.2	0.000	0.125	0.068	0.000	0.001	0.213	0.207	0.441	0.179	0.728
	.9/.2	0.000	0.126	0.383	0.000	0.000	0.555	0.199	0.913	0.033	1.135
	.9/.9	0.281	0.128	1.132	0.019	0.120	0.748	0.613	0.933	∞	∞
GSS		0.000	0.868	0.553	0.000	0.000	0.277	0.253	0.012	–	–
GSS-R		0.000	0.860	0.537	0.000	0.000	0.282	0.223	0.016	–	–
NM		0.012	0.088	1.265	0.166	0.083	0.635	1.331	0.692	–	–
NM-R		0.010	0.087	1.145	0.156	0.050	0.571	0.988	0.673	–	–
PSO	–0.5/2	0.018	0.300	0.493	0.000	0.002	0.257	0.202	0.562	0.020	0.000
	1/2	1.681	0.297	2.234	13.294	3.635	1.314	1.363	0.412	1.701	0.000
REA-P	R1	0.595	0.859	1.432	0.181	0.369	0.798	0.753	0.953	0.558	0.969
	R2	0.567	0.833	1.463	0.171	0.381	0.838	0.744	0.945	0.630	1.029
REA-T	R1	0.000	0.136	0.074	0.000	0.000	0.124	0.058	0.444	0.000	0.283
	R2	0.528	0.823	1.369	0.113	0.311	0.805	0.740	0.943	0.548	0.920
	R3	0.508	0.816	1.367	0.107	0.292	0.798	0.736	0.939	0.527	0.818
SA		0.002	0.778	0.822	0.000	0.000	0.330	0.321	0.901	0.009	0.628
rBOA		0.000	0.236	0.246	0.000	0.000	0.432	0.041	1.178	0.006	1.168
rGA		0.001	0.476	0.576	0.000	0.000	0.282	0.140	0.300	0.008	0.754

Table A.44: Variance for performance criterion ζ_{T_m} (scaled) with $m = 100,000$ in 25 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	0.074	0.356	0.019	0.187	0.139	0.277	0.285	0.000	0.112
CG-R		0.000	0.056	0.157	0.000	0.000	0.050	0.033	0.125	0.000	0.045
CMA-ES	100	0.000	0.008	0.010	0.000	0.000	0.009	0.010	0.097	0.000	∞
	750	0.000	0.002	0.005	0.000	0.000	0.001	0.007	0.092	0.000	∞
	1250	0.000	0.000	0.004	0.000	0.000	0.000	0.005	0.065	0.000	∞
	2500	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.035	0.000	∞
CMA-ES-R	100	0.000	0.002	0.040	0.015	0.006	0.064	0.014	96.977	0.000	∞
	750	0.000	0.000	0.008	0.015	0.008	0.001	0.006	82.606	0.000	∞
	1250	0.030	0.485	14.207	0.068	0.458	6.239	0.013	228.087	0.000	∞
	2500	0.115	0.727	32.238	0.109	2.577	36.096	0.084	299.574	0.000	∞
DE	.2/.9	0.030	0.014	0.075	0.001	0.012	0.043	0.055	0.050	0.239	0.063
	.2/.2	0.000	0.013	0.023	0.000	0.000	0.042	0.040	0.068	0.043	0.054
	.9/.2	0.000	0.013	0.067	0.000	0.000	0.047	0.033	0.055	0.053	0.045
	.9/.9	0.071	0.014	0.111	0.017	0.050	0.048	0.069	0.050	∞	∞
GSS		0.000	0.156	0.115	0.000	0.000	0.089	0.090	0.061	–	–
GSS-R		0.000	0.153	0.102	0.000	0.000	0.111	0.061	0.087	–	–
NM		0.056	0.024	0.404	0.676	0.253	0.146	0.145	0.100	–	–
NM-R		0.049	0.025	0.382	0.598	0.170	0.145	0.082	0.096	–	–
PSO	–0.5/2	0.018	0.014	0.132	0.000	0.002	0.134	0.086	0.137	0.016	0.000
	1/2	0.409	0.014	0.230	4.117	0.738	0.094	0.088	0.023	0.374	0.000
REA-P	R1	0.064	0.030	0.061	0.099	0.100	0.042	0.042	0.033	0.072	0.040
	R2	0.070	0.028	0.066	0.086	0.096	0.040	0.042	0.034	0.062	0.032
REA-T	R1	0.000	0.033	0.024	0.000	0.000	0.032	0.010	0.044	0.000	0.061
	R2	0.050	0.029	0.066	0.041	0.056	0.043	0.044	0.037	0.064	0.032
	R3	0.052	0.029	0.060	0.042	0.057	0.043	0.040	0.037	0.058	0.047
SA		0.000	0.057	0.078	0.000	0.000	0.068	0.050	0.049	0.000	0.072
rBOA		0.000	0.026	0.027	0.000	0.000	0.071	0.009	0.063	0.004	0.047
rGA		0.000	0.056	0.124	0.000	0.000	0.042	0.011	0.063	0.000	0.025

Table A.45: Results on performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 25 dimensions.

ζ_{T_m} (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	1.050	2.313	0.001	0.030	0.793	1.428	1.141	0.000	1.454
CG-R		0.000	0.794	1.473	0.000	0.000	0.417	0.007	0.583	0.000	1.110
CMA-ES	100	0.000	0.032	0.030	0.000	0.000	0.029	0.058	0.805	0.000	∞
	750	0.000	0.000	0.005	0.000	0.000	0.001	0.028	0.577	0.000	∞
	1250	0.000	0.000	0.003	0.000	0.000	0.000	0.020	0.530	0.000	∞
	2500	0.000	0.000	0.000	0.000	0.000	0.000	0.017	0.672	0.000	∞
CMA-ES-R	100	0.000	0.014	0.009	0.000	0.000	0.011	0.044	0.535	0.000	∞
	750	0.000	0.002	0.002	0.000	0.000	0.000	0.031	0.472	0.000	∞
	1250	0.016	0.289	0.268	0.000	0.002	0.138	0.046	0.772	0.000	∞
	2500	0.034	0.355	0.412	0.000	0.006	0.342	0.117	0.879	0.000	∞
DE	.2/.9	0.066	0.102	0.534	0.000	0.014	0.261	0.402	0.316	1.596	1.242
	.2/.2	0.000	0.103	0.043	0.000	0.000	0.082	0.136	0.193	0.027	0.623
	.9/.2	0.000	0.104	0.131	0.000	0.000	0.487	0.120	0.847	0.001	1.099
	.9/.9	0.204	0.104	1.017	0.007	0.070	0.704	0.536	0.894	∞	∞
GSS		0.000	0.868	0.553	0.000	0.000	0.277	0.253	0.012	–	–
GSS-R		0.000	0.132	0.480	0.000	0.000	0.230	0.188	0.008	–	–
NM		0.004	0.088	1.264	0.046	0.032	0.634	1.331	0.680	–	–
NM-R		0.004	0.087	0.996	0.050	0.021	0.502	0.934	0.637	–	–
PSO	–0.5/2	0.006	0.300	0.375	0.000	0.001	0.147	0.133	0.507	0.009	0.000
	1/2	1.623	0.297	2.232	13.294	3.635	1.313	1.363	0.411	1.651	0.000
REA-P	R1	0.524	0.836	1.372	0.168	0.352	0.759	0.700	0.929	0.479	0.862
	R2	0.544	0.800	1.411	0.159	0.363	0.807	0.693	0.912	0.582	1.009
REA-T	R1	0.000	0.082	0.073	0.000	0.000	0.123	0.058	0.424	0.000	0.283
	R2	0.463	0.791	1.305	0.071	0.238	0.769	0.692	0.914	0.483	0.803
	R3	0.441	0.786	1.295	0.065	0.229	0.760	0.692	0.905	0.464	0.557
SA		0.002	0.709	0.746	0.000	0.000	0.286	0.204	0.857	0.008	0.550
rBOA		0.000	0.208	0.204	0.000	0.000	0.358	0.027	1.178	0.001	1.165
rGA		0.001	0.466	0.576	0.000	0.000	0.250	0.133	0.296	0.008	0.738

Table A.46: Variance for performance criterion ζ_{T_m} (scaled) with $m = 250,000$ in 25 dimensions.

$Var(\zeta_{T_m})$ (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	0.074	0.356	0.019	0.187	0.139	0.277	0.285	0.000	0.112
CG-R		0.000	0.054	0.138	0.000	0.000	0.045	0.011	0.069	0.000	0.046
CMA-ES	100	0.000	0.008	0.010	0.000	0.000	0.009	0.010	0.097	0.000	∞
	750	0.000	0.002	0.005	0.000	0.000	0.001	0.007	0.087	0.000	∞
	1250	0.000	0.000	0.004	0.000	0.000	0.000	0.005	0.076	0.000	∞
	2500	0.000	0.000	0.002	0.000	0.000	0.000	0.000	0.134	0.000	∞
CMA-ES-R	100	0.000	0.001	0.019	0.014	0.006	0.042	0.011	78.474	0.000	∞
	750	0.000	0.000	0.002	0.015	0.008	0.000	0.006	61.078	0.000	∞
	1250	0.024	0.462	13.484	0.065	0.400	5.286	0.012	162.192	0.000	∞
	2500	0.104	0.696	31.644	0.096	2.476	32.253	0.080	209.650	0.000	∞
DE	.2/.9	0.016	0.011	0.054	0.000	0.004	0.035	0.047	0.045	0.249	0.067
	.2/.2	0.000	0.012	0.022	0.000	0.000	0.033	0.034	0.065	0.036	0.050
	.9/.2	0.000	0.011	0.064	0.000	0.000	0.052	0.023	0.061	0.000	0.048
	.9/.9	0.056	0.011	0.101	0.007	0.032	0.058	0.064	0.053	∞	∞
GSS		0.000	0.156	0.115	0.000	0.000	0.089	0.090	0.061	–	–
GSS-R		0.000	0.019	0.069	0.000	0.000	0.050	0.041	0.076	–	–
NM		0.033	0.024	0.403	0.224	0.145	0.146	0.145	0.097	–	–
NM-R		0.031	0.025	0.293	0.282	0.110	0.121	0.075	0.083	–	–
PSO	–0.5/2	0.005	0.014	0.102	0.000	0.000	0.078	0.054	0.135	0.005	0.000
	1/2	0.466	0.014	0.232	4.117	0.738	0.097	0.088	0.021	0.437	0.000
REA-P	R1	0.061	0.027	0.063	0.096	0.104	0.042	0.041	0.032	0.052	0.070
	R2	0.083	0.029	0.064	0.088	0.101	0.036	0.047	0.038	0.062	0.027
REA-T	R1	0.000	0.014	0.024	0.000	0.000	0.032	0.010	0.044	0.000	0.061
	R2	0.048	0.026	0.058	0.027	0.048	0.038	0.037	0.037	0.052	0.040
	R3	0.047	0.026	0.056	0.023	0.043	0.041	0.038	0.035	0.051	0.102
SA		0.000	0.057	0.066	0.000	0.000	0.065	0.029	0.051	0.000	0.080
rBOA		0.000	0.026	0.023	0.000	0.000	0.064	0.007	0.063	0.001	0.045
rGA		0.000	0.056	0.124	0.000	0.000	0.037	0.010	0.063	0.000	0.026

Table A.47: Results on performance criterion ϕ_1 (scaled) in 25 dimensions.

ϕ_1 (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	1.050	2.316	0.012	0.127	0.793	1.429	1.148	0.000	1.454
CG-R		0.000	0.829	1.617	0.000	0.000	0.464	0.069	0.687	0.000	∞
CMA-ES	100	0.000	0.036	0.034	0.000	0.000	0.036	0.059	0.814	0.000	∞
	750	0.013	0.039	0.060	0.002	0.008	0.053	0.051	0.733	0.010	∞
	1250	0.026	0.067	0.098	0.005	0.015	0.086	0.063	0.819	0.021	∞
	2500	0.054	0.139	0.198	0.008	0.029	0.169	0.107	0.997	0.049	∞
CMA-ES-R	100	0.000	0.023	0.018	0.000	0.000	0.022	0.050	0.614	0.000	∞
	750	0.003	0.019	0.027	0.000	0.002	0.024	0.041	0.594	0.004	∞
	1250	0.039	0.320	0.328	0.003	0.014	0.206	0.083	0.903	0.001	∞
	2500	0.084	0.406	0.511	0.008	0.034	0.421	0.186	1.013	0.000	∞
DE	.2/.9	0.149	0.128	0.726	0.015	0.059	0.374	0.507	0.458	1.641	1.252
	.2/.2	0.024	0.129	0.174	0.007	0.015	0.229	0.241	0.417	0.197	0.738
	.9/.2	0.019	0.129	0.389	0.001	0.008	0.559	0.218	0.910	0.106	1.135
	.9/.9	0.295	0.130	1.132	0.033	0.137	0.751	0.614	0.933	∞	∞
GSS		0.009	0.874	0.565	0.000	0.002	0.281	0.256	0.025	0.041	0.066
GSS-R		0.009	0.547	0.526	0.000	0.003	0.264	0.228	0.026	0.040	0.060
NM		0.017	0.088	1.265	0.132	0.073	0.635	1.331	0.692	0.253	0.973
NM-R		0.015	0.087	1.121	0.150	0.044	0.559	0.994	0.668	0.230	0.956
PSO	−0.5/2	0.034	0.300	0.520	0.001	0.009	0.290	0.219	0.576	0.035	0.000
	1/2	1.686	0.297	2.240	13.294	3.636	1.314	1.363	0.414	1.705	0.000
REA-P	R1	0.600	0.861	1.436	0.197	0.385	0.804	0.753	0.957	0.559	0.949
	R2	0.577	0.833	1.472	0.187	0.395	0.842	0.750	0.946	0.636	1.029
REA-T	R1	0.002	0.162	0.138	0.000	0.001	0.164	0.069	0.502	0.005	0.293
	R2	0.537	0.825	1.377	0.135	0.323	0.811	0.741	0.944	0.555	0.908
	R3	0.515	0.819	1.368	0.126	0.305	0.801	0.739	0.940	0.535	0.784
SA		0.003	0.778	0.839	0.000	0.000	0.349	0.357	0.901	0.009	0.637
rBOA		0.015	0.239	0.254	0.006	0.009	0.446	0.049	1.178	0.019	1.165
rGA		0.009	0.479	0.585	0.002	0.005	0.283	0.159	0.305	0.014	0.757

Table A.48: Variance for performance criterion ϕ_1 (scaled) in 25 dimensions.

$Var(\phi_1)$ (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	0.074	0.356	0.019	0.176	0.139	0.277	0.283	0.000	0.112
CG-R		0.000	0.046	0.120	0.000	0.000	0.038	0.041	0.085	0.000	∞
CMA-ES	100	0.000	0.008	0.010	0.000	0.000	0.009	0.010	0.094	0.000	∞
	750	0.001	0.002	0.005	0.001	0.001	0.003	0.006	0.064	0.001	∞
	1250	0.002	0.001	0.004	0.002	0.002	0.002	0.004	0.048	0.001	∞
	2500	0.005	0.002	0.005	0.002	0.005	0.005	0.003	0.034	0.003	∞
CMA-ES-R	100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	750	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1250	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	2500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DE	.2/.9	0.020	0.010	0.052	0.005	0.010	0.028	0.035	0.035	0.233	0.061
	.2/.2	0.002	0.010	0.021	0.002	0.002	0.029	0.033	0.057	0.038	0.040
	.9/.2	0.002	0.010	0.050	0.000	0.002	0.034	0.022	0.043	0.031	0.038
	.9/.9	0.053	0.010	0.086	0.016	0.037	0.042	0.055	0.039	∞	∞
GSS		0.001	0.153	0.114	0.000	0.001	0.089	0.089	0.060	0.008	0.063
GSS-R		0.002	0.073	0.076	0.000	0.001	0.073	0.054	0.076	0.007	0.062
NM		0.051	0.024	0.404	0.490	0.222	0.146	0.145	0.098	0.207	0.261
NM-R		0.045	0.025	0.330	0.524	0.149	0.120	0.063	0.088	0.196	0.253
PSO	–0.5/2	0.016	0.014	0.099	0.001	0.005	0.091	0.060	0.123	0.014	0.000
	1/2	0.393	0.014	0.216	4.117	0.738	0.091	0.088	0.023	0.358	0.000
REA-P	R1	0.046	0.022	0.048	0.094	0.093	0.031	0.030	0.026	0.049	0.035
	R2	0.065	0.021	0.051	0.082	0.090	0.030	0.032	0.028	0.049	0.025
REA-T	R1	0.000	0.015	0.021	0.000	0.000	0.030	0.009	0.038	0.000	0.060
	R2	0.036	0.020	0.049	0.030	0.038	0.031	0.030	0.028	0.044	0.020
	R3	0.037	0.022	0.046	0.027	0.039	0.032	0.028	0.027	0.037	0.048
SA		0.000	0.044	0.058	0.000	0.000	0.054	0.038	0.039	0.000	0.062
rBOA		0.005	0.021	0.020	0.004	0.003	0.051	0.005	0.063	0.005	0.045
rGA		0.002	0.056	0.123	0.001	0.001	0.037	0.010	0.063	0.001	0.020

Table A.49: Results on performance criterion ϕ_2 (scaled) in 25 dimensions.

ϕ_2 (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	1.078	2.705	0.274	1.202	0.890	1.449	1.734	0.001	1.467
CG-R		0.000	1.076	2.777	0.000	0.000	0.977	1.211	1.060	0.002	∞
CMA-ES	100	0.093	0.501	0.733	0.000	0.024	0.681	0.361	1.214	0.079	∞
	750	0.640	0.847	1.426	0.237	0.462	0.902	0.783	1.201	0.583	∞
	1250	0.635	0.856	1.445	0.228	0.460	0.907	0.800	1.206	0.595	∞
	2500	0.519	0.814	1.341	0.105	0.313	0.825	0.726	1.146	0.582	∞
CMA-ES-R	100	0.075	0.485	0.692	0.000	0.018	0.670	0.335	1.185	0.067	∞
	750	0.530	0.778	1.290	0.110	0.326	0.838	0.703	1.168	0.486	∞
	1250	0.547	0.823	1.378	0.139	0.347	0.851	0.747	1.162	0.363	∞
	2500	0.522	0.810	1.348	0.106	0.314	0.828	0.728	1.146	0.077	∞
DE	.2/.9	0.976	0.247	1.759	1.093	0.998	1.007	1.017	1.052	1.692	1.265
	.2/.2	0.954	0.248	1.723	0.943	0.957	0.994	1.007	1.053	1.161	1.134
	.9/.2	0.654	0.248	1.522	0.272	0.507	0.923	0.844	1.100	1.152	1.270
	.9/.9	0.834	0.248	1.650	0.568	0.774	0.977	0.929	1.103	∞	∞
GSS		0.865	1.137	1.612	0.330	0.522	0.793	0.760	0.786	0.849	0.777
GSS-R		0.829	1.143	1.615	0.337	0.566	0.793	0.763	0.797	0.838	0.779
NM		0.290	0.094	1.306	0.603	0.328	0.680	1.331	0.771	0.343	0.973
NM-R		0.302	0.092	1.284	0.717	0.302	0.686	1.348	0.761	0.315	0.966
PSO	–0.5/2	0.398	0.300	1.285	0.119	0.234	0.938	0.738	0.912	0.409	0.002
	1/2	1.842	0.298	2.284	13.307	3.651	1.321	1.366	0.666	1.828	0.000
REA-P	R1	0.998	0.995	1.765	0.870	0.896	1.018	0.995	1.094	0.973	1.127
	R2	0.905	0.993	1.772	0.833	0.900	1.027	0.993	1.089	0.975	1.128
REA-T	R1	0.394	0.849	1.436	0.051	0.199	0.892	0.726	1.092	0.534	0.936
	R2	0.943	0.984	1.740	0.936	0.950	1.020	0.978	1.100	0.960	1.117
	R3	0.928	0.984	1.740	0.918	0.916	1.012	0.983	1.091	0.945	1.117
SA		0.035	1.005	1.414	0.190	0.002	0.893	1.073	1.081	0.019	1.058
rBOA		0.830	0.402	0.554	1.021	0.893	1.068	0.219	1.178	1.032	1.169
rGA		0.673	0.874	1.446	0.541	0.600	0.813	0.894	0.836	0.671	0.976

Table A.50: Variance for performance criterion ϕ_2 (scaled) in 25 dimensions.

$Var(\phi_2)$ (scaled)		sphr.	ack.	lg-ack.	whit.	rosen.	rastr.	sal.	schw.	grie.	weier.
CG		0.000	0.098	0.551	0.000	0.000	0.195	0.298	0.000	–	0.124
CG-R		0.000	0.106	0.519	–	–	0.262	0.394	0.000	–	∞
CMA-ES	100	0.022	0.024	0.057	0.000	0.009	0.046	0.044	0.043	0.020	∞
	750	0.080	0.037	0.081	0.107	0.114	0.061	0.057	0.041	0.073	∞
	1250	0.079	0.034	0.077	0.102	0.102	0.050	0.044	0.038	0.074	∞
	2500	0.064	0.031	0.073	0.045	0.066	0.046	0.047	0.039	0.072	∞
CMA-ES-R	100	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	750	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1250	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	2500	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
DE	.2/.9	0.124	0.014	0.115	0.489	0.251	0.069	0.066	0.055	0.230	0.058
	.2/.2	0.123	0.015	0.110	0.375	0.226	0.070	0.063	0.058	0.153	0.050
	.9/.2	0.111	0.015	0.111	0.148	0.150	0.056	0.065	0.045	0.199	0.045
	.9/.9	0.105	0.015	0.094	0.245	0.164	0.049	0.052	0.047	∞	∞
GSS		0.154	0.088	0.150	0.260	0.210	0.108	0.104	0.133	0.165	0.088
GSS-R		0.161	0.084	0.140	0.260	0.235	0.123	0.101	0.135	0.151	0.083
NM		0.230	0.025	0.414	1.220	0.518	0.158	0.145	0.099	0.244	0.261
NM-R		0.212	0.025	0.406	1.349	0.381	0.165	0.148	0.095	0.223	0.258
PSO	–0.5/2	0.151	0.014	0.170	0.201	0.141	0.090	0.111	0.096	0.152	0.007
	1/2	0.256	0.014	0.146	4.109	0.720	0.080	0.085	0.135	0.220	0.000
REA-P	R1	0.099	0.032	0.085	0.366	0.203	0.055	0.052	0.043	0.108	0.041
	R2	0.098	0.037	0.077	0.320	0.185	0.053	0.056	0.048	0.104	0.044
REA-T	R1	0.057	0.030	0.075	0.033	0.057	0.052	0.055	0.043	0.065	0.046
	R2	0.104	0.033	0.086	0.352	0.190	0.057	0.056	0.043	0.106	0.042
	R3	0.094	0.037	0.079	0.323	0.198	0.056	0.055	0.045	0.109	0.039
SA		0.010	0.044	0.125	0.254	0.001	0.119	0.076	0.043	0.003	0.075
rBOA		0.221	0.051	0.101	0.712	0.370	0.100	0.031	0.063	0.227	0.048
rGA		0.107	0.042	0.113	0.272	0.164	0.066	0.071	0.055	0.105	0.040

Table A.51: Results of neural network experiments using the performance criteria of Chapter 8: success probability (σ_ϵ^N), hitting time on success ($\hat{\psi}_\epsilon^N$), final error (ζ_T), average error (ϕ_1), and weighted average error (ϕ_2). The first standard deviation is provided where possible; the value for σ_ϵ^N is accurate within ± 0.005 with $p < 0.05$. The error threshold ϵ was chosen separately for each task and is given in the table. For Currency Trading, raw fitness values are shown instead of errors. Neuroannealing outperforms NEAT on the Multiplexers and Concentric Spirals.

Neuroannealing

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Double Pole (Markov)	1.0	0.845	7,767 ± 4,871	0.154 ± 0.360	0.203 ± 0.342	0.991 ± 0.071
Double Pole (non-Markov)	1.0	0.960	7,499 ± 3,157	0.039 ± 0.195	0.163 ± 0.181	0.998 ± 0.006
Multiplexer, 1 × 2	0.010	0.964	11,278 ± 3,781	0.004 ± 0.023	0.030 ± 0.020	0.208 ± 0.045
Multiplexer, 2 × 4	0.200	0.047	19,833 ± 10,351	0.247 ± 0.037	0.252 ± 0.025	0.329 ± 0.023
Multiplexer, 3 × 5	0.250	0.028	20,566 ± 15,509	0.285 ± 0.013	0.287 ± 0.013	0.363 ± 0.017
Multiplexer, 3 × 6	0.300	0.036	17,675 ± 12,449	0.305 ± 0.013	0.308 ± 0.011	0.385 ± 0.012
Concentric Spirals	0.300	0.261	21,687 ± 7,834	0.310 ± 0.021	0.317 ± 0.014	0.333 ± 0.001
Currency Trading	$f > 250$	0.749	20,054 ± 11,189	31016.331 ± 55094.212	10904.020 ± 20980.836	0.930 ± 0.284

NEAT

Task	$\epsilon <$	σ_ϵ^N	$\hat{\psi}_\epsilon^N$	ζ_T	ϕ_1	ϕ_2
Double Pole (Markov)	1.0	1.000	1,819 ± 2,276	0.000 ± 0.000	0.018 ± 0.044	0.566 ± 0.472
Double Pole (non-Markov)	1.0	1.000	4,676 ± 2,107	0.000 ± 0.000	0.012 ± 0.012	0.742 ± 0.415
Multiplexer, 1 × 2	0.010	0.000	50,000 ± 0,000	0.166 ± 0.027	0.180 ± 0.012	0.187 ± 0.000
Multiplexer, 2 × 4	0.200	0.000	50,000 ± 0,000	0.279 ± 0.001	0.282 ± 0.001	0.300 ± 0.008
Multiplexer, 3 × 5	0.250	0.000	50,000 ± 0,000	0.322 ± 0.001	0.325 ± 0.001	0.340 ± 0.007
Multiplexer, 3 × 6	0.300	0.000	50,000 ± 0,000	0.348 ± 0.003	0.351 ± 0.002	0.367 ± 0.005
Concentric Spirals	0.300	0.000	50,000 ± 0,000	0.331 ± 0.000	0.331 ± 0.000	0.332 ± 0.000
Currency Trading	$f > 250$	0.028	29,425 ± 8,095	43.365 ± 103.898	15.910 ± 48.626	0.033 ± 0.105

Appendix B

Automated Currency Exchange Trading

This appendix describes the automated currency exchange trading task for the experiments in Chapter 13, including the relevant background, datasets, and input preprocessing.

B.1 Currency Trading Background

The exchange rates of currencies that are freely floated present a stochastic sequence that is difficult to predict. Speculators attempt to make money by buying and selling currencies at high leverage. Most of these forex traders lose money, but a few consistently average a profit, suggesting that there is some structure to the problem that can be learned. The advantage to the domain is that there are relatively few inputs and outputs, allowing for fast prototyping and exploration. The high stochasticity of the signal is challenging, especially in light of the relevance of exogenous information, such as Central Bank announcements that cannot easily be modeled in a numeric setting.

Trading currencies bears substantial similarities to trading stocks in terms of market function and techniques. Most speculative trading takes place through brokers on exchanges that are separate from retail currency exchange markets; speculation is generally regarded as playing the role of price discovery for other markets. Speculative traders primarily trade with each other.

The focus of this study is short-term trading using Technical Analysis [108, 145, 161]. Technical Analysis is an approach to trading that is characterized by the mechanistic use of a set of statistics termed *technical indicators*. These technical indicators can be computed for any price sequence but are considered more effective or meaningful in the context of markets with high liquidity and large trading volumes, two features that are preeminently characteristic of foreign exchange (or *forex*) markets.

Price in technical trading is typically broken up into segments, the size of which depends on the desired frequency of trading. Within each period, four prices are recorded, specifically the high, low, opening and closing price. The closing values are used to compute technical indicators. Typical technical indicators include various averages, Stochastics, Relative Strength Index (RSI), Bollinger Bands, and Fibonacci levels [108]. The Simple Moving Average (SMA) is an unweighted average over a fixed number of periods starting from the current period. The Exponential Moving Average (EMA) is defined by a discrete update rule, $\mathrm{EMA}(t+1) = \alpha\mathrm{EMA}(t) + (1-\alpha)\mathrm{Close}(t)$, where $0 \leq \alpha \leq 1$; the EMA is said to be taken over N periods where $N = \frac{2}{\alpha} - 1$. A common use of moving averages is to identify trends by examining the ratio between two moving averages computed from different periods, with the average over a shorter period on top [161]. If the ratio is greater than 1, then the price is concluded to be in an uptrend since the older data is less than newer data on average. If the ratio is less than 1, then the price is in a downtrend.

Stochastics (Stoch) and Relative Strength Index (RSI) are oscillators that vary between 0 and 100. The Stochastic is given by

$$\mathrm{Stoch}(t) = 100\frac{\mathrm{Close}(t) - \mathrm{Low}(t)}{\mathrm{High}(t) - \mathrm{Low}(t)}$$

where the high, low and close are taken over all periods involved, e.g. a three period stochastic takes the high and low as the extrema of the prior 3 periods and the close from the last period [145]. The Stochastic encodes whether the current price (the most recent close) is high or low relative to recent history. Because the stochastic focuses on recent history, it tends to change faster than EMA ratios. It is also possible to compute a Stochastic from the EMA instead of the price; this is called the Slow Stochastic. As with moving averages, ratios of fast and slow Stochastics as well as ratios between Stochastics of different periods are used as trend change indicators. Stochastics also tend to exhibit a behavior termed *divergence* near price peaks. *Negative divergence* occurs when the Stochastic makes a peak above 80 and then makes a secondary peak at a lower value while the price makes a new high; this behavior often indicates a coming shift in trend to a bear market and can be used as a trading signal [108]. *Positive divergence* refers to the reverse situation with a trough below 20, and indicates a change to a bull market. RSI is defined differently, but tends to exhibit similar behavior.

Bollinger bands are a pair of lines placed around the price above and below at a distance from the price on either side equal to the standard deviation of the price from its average over a fixed number of preceding periods [161]. Prices can tend to reverse after penetrating the Bollinger bands on either side. Additionally, when the Bollinger bands contract due to a reduction in price variation, it generally indicates a period of volatility to follow.

A currency trader makes decisions consisting of orders to buy or sell currencies. Currencies are expressed as pairs, e.g. EUR/USD. For EUR/USD, the euro is the *base currency*, and the US dollar is the *counter currency.* An order has a type and an associated value. The order type can be BUY, SELL, SELL SHORT or BUY TO COVER. In a BUY order, the counter currency is traded for the base currency in the amount specified; once executed, the trader is said to have entered a long position. A SELL order exits all or a portion of a long positions depending on the value. A SELL SHORT order borrows the base currency in the amount specified in order to buy the counter currency, thereby entering a short position. A BUY TO COVER order exits all or part of an existing short position, repaying the borrowed money. The value of an order is expressed in *lots*, denominated in US dollars for the purpose of this research. A standard lot is US$100,000; a *mini-lot* is worth US$ 10,000. When a trade is entered where neither currency is USD, the actual currency amount purchased or sold is determined based on the current exchange rate with the US dollar.

Currency traders are allowed to purchase an amount up to 100 times the current value of the trader's account. The actual multiple of the account value currently held in long or short positions is termed *leverage.* After trades are entered, then the trader is allowed to hold the position until the account value dwindles to the *margin*, usually at 200 times the value of the trader's account. At that point, the positions are liquidated by the brokerage; this is termed a *margin call.* As long as the value of the account remains above the margin, the trader has discretion as to when the trade should end.

Orders can be executed by several means. A *market order* queues up for execution at the current exchange rate, which may fluctuate prior to execution, since trades must execute in order. A *limit order* specifies a maximum exchange rate for long trades, and will only execute if the price falls to or below the maximum; for short trades, the limit specifies a minimum rather than a maximum. Limit orders are given preference to market orders for execution. Finally, a *stop loss order*, or simply *stop* is the mirror image of a limit order; it specifies a maximum exchange rate for executing a short trade, or a minimum exchange rate for executing a long trade. Stops are used to protect against catastrophic loss on a trade. Several other order types are available that will not be used in this research.

B.2 Currency Trading Experiments

Automated currency trading was used to verify the application of evolutionary annealing to RNNs experimentally. The task of currency trading was described in Section B.1 along with applicable technical indicators. It is an advantage of the currency trading domain that it involves relatively few inputs and outputs while still providing a difficult task where success is subject to uncertainty. These aspects make the currency domain a solid testbed for experimental verification of neuroannealing. This section describes how neural network controllers can be evaluated for the currency trading task.

B.2.1 Experimental Setup

The task of an automated currency trader is to progressively read a sequence of technical indicators and output trading decisions. The sequence of technical indicators for this experiment consists of ten real-valued inputs derived from the exponential moving average at five, 20, and 50 periods (EMA-5, EMA-20, EMA-50), the relative strength index at 14 periods (RSI), the fast and slow stochastics at 14 and three periods respectively (FSTOCH and SSTOCH), the width of the Bollinger Bands (BB-W), the position of the closing price within the Bollinger Bands (BB-P), the absolute difference between the opening and closing price (OC), and the difference between the high and low price (HL). All of these indicators are commonly used and definitions can be readily found in any materials on the subject of Technical Analysis [108, 145, 161].

In order to train currency traders on multiple currency pairs, it is necessary to remove any reference to the absolute magnitude of the price. Three inputs are used for the ratio of price to EMA-5, EMA-20, and EMA-50, respectively. One input each is used for RSI, FSTOCH, and SSTOCH, since these indicators are already independent of price. Two more inputs are used for BB-P and the log ratio of BB-W to its exponential moving average. The final two inputs include the log ratio of OC and HL to their respective moving averages. Inputs are centered to have both positive and negative values and scaled to approximately the same order of magnitude.

For these experiment, several simplifying assumptions regarding trading are made. During each training and testing run, a trader trades a single currency pair at a time. Whenever the trader has no position in the currency, then the trader can issue BUY or SHORT decisions to enter a trade in a long or short position respectively; alternately, the trader may WAIT and do nothing. Leverage is determined by the strength of the BUY signal versus the SHORT signal or vice versa. Once a position is entered, the trader may either EXIT the position or HOLD it at each time step; the trader may also increase or decrease the leverage on the position. Positions are entered with a limit order fixed to the close of the prior period. In the simulation, all of these limit orders succeed, which is realistic if the limit order is issued substantially close in time to the closing price. Stop loss limits are not used in these simulations to simplify the problem. In a practical implementation, a large stop could be entered to prevent catastrophic loss, representing the portion of the account value to be placed at risk during the next hour. These assumptions are intended to give structure to the experiment and do not significantly restrict the generality of the task. However, these choices do exclude some trading strategies, including arbitrage trades, where a sequence of trades involving at least three currencies exploits short-term imbalances among currencies, and staged entry approaches where the position size is increased at set points if the trade proceeds as expected.

The goal of currency trading is to maximize gain during a trading run. A secondary goal is to simultaneously minimize risk, typically volatility. Thus a trader that steadily increases its account value without substantial losses during the run is preferable over a trader that increases the account value very quickly, but at the cost of substantial volatility and drawdown. In the current experiments, the objective value of a trader was measured solely based on the final account value without taking risk into account specifically. Future implementations may explore a multi-objective setting in which gain must be maximized while minimizing risk over any period.

Training takes place in a simulated trading environment using a fixed data set described below. Since the data set consists only of hourly trading data, it is not possible to tell during simulation whether a limit or a stop would have been executed if both prices were reached in the same trading period. For the purpose of simulation, limits were always executed and stops were not used. This decision has the effect of potentially inflating gains, but as such it affects all the trading networks evenly. As mentioned above, the assumption that limits execute is realistic if the network makes decisions in real time at the close of each period.

B.2.2 Currency Exchange Rate Data Set

The available training data consists of six months of hourly trading data from September 2009 to February 2010 on 18 separate currency pairs, obtained from a commercial brokerage. These include the six forex majors, EUR/USD, AUD/USD, USD/CAD, GBP/USD, USD/JPY, USD/CHF, as well as twelve other currency crosses, EUR/JPY, EUR/GBP, EUR/AUD, GBP/JPY, GBP/CHF, CHF/JPY, CAD/JPY, AUD/JPY, NZD/USD, AUD/NZD, AUD/CAD, and AUD/CHF.

The neural networks in the experiments are tested on a subset of nine of these trading sets: EUR/USD, GBP/USD, USD/CHF, USD/JPY, NZD/USD, USD/CAD, AUD/CAD, and AUD/NZD. These pairs are interesting since all but two of them include the US dollar. The further simplifying assumption is made that the trading account is denominated in the counter currency. In a practical implementation, the trading account would be denominated in a single currency, likely in US dollars.

The use of a dataset from a single time period incurs a risk that the dataset contains internal correlations that would not be reflected during other time periods. This problem is mitigated somewhat by including a variety of currencies from countries with substantially different economies, and by the inclusion of two pairs without the US dollar. Also, since trading is performed at an hourly scale, the use of six months of data (over $3,000$ hours) means that a large variety of trading situations are encountered.

There is also a risk that an automated trader trained on a particular dataset will overfit the data, learning a trading strategy that only works on this particular dataset. In some sense, this risk exists no matter what data is used. The presumption that the past is predictive of the future underlies all forms of learning. In this particular case, the use of fixed technical indicators should hide any specific price cues that are not also of general use as trading signals. Hopefully, simply memorizing the best output for each inputs should be a dangerous strategy given $27,000$ hours of trading on distinct currency pairs. It is unknown at this time whether memorizing the data is a viable strategy. Even if so, in the current context, this experiment is simply treated as an objective function to maximize, which is of value for assessing the learning abilities of neuroannealing in any case.

Bibliography

[1] Ackley, Hinton, and Sejnowski. A learning algorithm for boltzmann machines. *Cognitive Science*, 9:147–169, 1985.

[2] David H. Ackley. *A connectionist machine for genetic hillclimbing.* Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[3] Chang Ahn, R. Ramakrishna, and David Goldberg. Real-coded bayesian optimization algorithm. In Jose Lozano, Pedro Larrañaga, I. Inza, and Endika Bengoetxea, editors, *Towards a New Evolutionary Computation*, volume 192 of *Studies in Fuzziness and Soft Computing*, pages 51–73. Springer Berlin / Heidelberg, 2006.

[4] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6), 1974.

[5] Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi. Bidirectional relation between cma evolution strategies and natural evolution strategies. *Parallel Problem Solving from Nature, PPSN XI*, 2010.

[6] Matthew Alden. *MARLEDA: Effective Distribution Estimation Through Markov Random Fields.* PhD thesis, Department of Computer Sciences, the University of Texas at Austin, Austin, Texas, 2007. Also Technical Report AI07-349.

[7] M. Montaz Ali, Charoenchai Khompatraporn, and Zelda B. Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31:635–672, 2005. 10.1007/s10898-004-9972-2.

[8] Charalambos C. Aliprantis and Kim D. Border. *Infinite Dimensional Analysis: A Hitchhiker's Guide, 3rd Edition.* Springer, New York, New York, 2006.

[9] L. Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16, 1966.

[10] D. Ashlock. Taxonomic clustering of genetic algorithms using unique performance signatures. private communication, 2011.

[11] C. Audet and J.E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.*, 17, 2006.

[12] A. Auger and O. Teytaud. Continuous lunches are free! In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO-2007)*, New York, 2007. ACM Press.

[13] Anne Auger and Niklaus Hansen. A restart cma evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, 2005.

[14] T. Bäck, F. Hoffmeister, and H. P. Schwefel. A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, California, 1991. Morgan Kauffman.

[15] J. D. Bagley. *The Behavior of Adaptive Systems Which Employ Genetic and Correlation Algorithms*. PhD thesis, University of Michigan, Ann Arbor, 1967.

[16] J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their Applications*, 1985.

[17] Shumeet Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[18] Shumeet Baluja and Rich Caruana. *Removing the Genetics from the Standard Genetic Algorithm*. Morgan Kaufmann, 1995.

[19] N. E. Barabanov and D. V. Prokhorov. Stability analysis of discrete-time recurrent neural networks. *IEEE Transactions on Neural Networks*, 13(2), 2002.

[20] N. Bard and M. Bowling. Particle filtering for dynamic agent modeling in simplified poker. In *Proceedings of the 22nd Conference on Artificial Intelligence*, Madison, WI, 2007. AAAI Press.

[21] L. Barone and L. While. Adaptive learning for poker. In *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, California, 2000. Kaufmann.

[22] Nils Aal Barricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, IX(35–36), 1957.

[23] Sterling K. Berberian. *Lectures in Functional Analysis and Operator Theory.* Springer-Verlag, New York, New York, 1974.

[24] H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. M. Gambardella. Results of the first international contest on evolutionary optimisation (1st iceo). In *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996.

[25] H. Bersini and F.J. Varela. Hints for adaptive problem solving gleaned from immune networks. In *Parallel Problem Solving from Nature, First Workshop*, Dortmund, Germany, 1990.

[26] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical Science*, 8(1), 1993.

[27] Hans-Georg Beyer. *Theory of Evolution Strategies.* Springer-Verlag, Berlin, Germany, 2001.

[28] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 2003 International Joint Conference on Artificial Intelligence*, 2003.

[29] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *Proceedings of 15th National Conference of the American Association on Artificial Intelligence*, Madison, WI, 1998. AAAI Press.

[30] Patrick Billingsley. *Probability and Measure.* John Wiley, 1986.

[31] R. J. C. Bosman, W. A. van Leeuwen, and B. Wemmenhove. Combining hebbian and reinforcement learning in a minibrain model. *Neural Netw.*, 17:29–36, January 2004.

[32] H. J. Bremerman. *Optimization through Evolution and Recombination.* Spartan Books, Washington, D.C, 1958.

[33] A. Brindle. *Genetic Algorithms for Function Optimization.* PhD thesis, University of Alberta, Edmonton, 1981.

[34] K. M. Bryden, D. A. Ashlock, S. Corns, and S. Willson. Graph-based evolutionary algorithms. *IEEE Transaction on Evolutionary Computation*, 10(5), 2006.

[35] J. Cabessa and H. T. Siegelman. Evolving recurrent neural networks are super-turing. In *Proceedings of the International Joint Conference on Neural Networks*, 2011.

[36] Gerolamo Cardano. *Liber de Ludo Aleae.* 1526.

[37] G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of Computing Machinery*, 22(3), 1975.

[38] K. L. Chung and R.J. Williams. *Introduction to Stochastic Integration.* Birkhauser, Boston, MA, 1990.

[39] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber. Deep big simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12), 2010.

[40] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *International Joint Conference on Artificial Intelligence (IJCAI-2011, Barcelona)*, 2011.

[41] D. C. Ciresan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks (IJCNN-2011, San Francisco)*, 2011.

[42] Jonathan Coens. *Taking Tekkotsu Out of the Plane.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 2010.

[43] Donald Cohn. *Measure Theory.* Birkhauser, Boston, MA, 1980.

[44] M. Colombetti and M. Colombetti. Learning to control an autonomous robot by distribution genetic algorithms. In *From Animals to Animats 2: Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior.* MIT Press, 1992.

[45] I. D. Coope and C. J. Price. On the convergence of grid-based methods for unconstrained optimization. *SIAM J. Optim.*, 11, 2001.

[46] A. Corduneanu and C. M. Bishop. Variational bayesian model selection for mixture distributions. In *In Proceedings of the Eighth International Conference on Articial Intelligence and Statistics.* Morgan Kaufmann, 2001.

[47] T. M. Cover. Universal gambling schemes and the complexity measures of kolmogorov and chaitin. Technical Report Rep. 12, Statistics Dept., Stanford University, 1974.

[48] J. C. Culberson. On the futility of blind search: an algorithmic view of “no free lunch”. *Evolutionary Computation*, 6(2), 1998.

[49] G. Cybenko. Approximation by superpositions of sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2, 1989.

[50] P. J. Daniell. Integrals in an infinite number of dimensions. *Annals of Mathematics*, 20, 1919.

[51] Georges A. Darbellay and Igor Vajda. Estimation of the information by an adaptive partitioning of the observation space. *IEEE Transactions on Information Theory*, pages 1315–1321, 1999.

[52] S. Das, A. Konar, and U.K. Chakraborty. Annealed differential evolution. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1926 –1933, sept. 2007.

[53] A. Davidson. Using artificial neural networks to model opponents in texas hold em. Technical Report http://spaz.ca/aaron/poker/nnpoker.pdf, Unpublished manuscript, 1999.

[54] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence*, 2000.

[55] Pierre de Fermat. *Methodus ad disquirendam maximam et minima.* 1638.

[56] Kenneth A. de Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[57] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B (Methodological)*, 39(1), 1977.

[58] Marco Dorigo. *Optimization, Learning and Natural Algorithms.* PhD thesis, Politecnico di Milano, 1992.

[59] S. Droste, T. Jansen, and I. Wegener. Optimization with randomized search heuristics – the (a)nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287:2002, 1997.

[60] S. Droste, T. Jansen, and I. Wegener. Perhaps not a free lunch but at least a free appetizer. Technical Report No. CI-45/98, University of Dortmund, 1998.

[61] W. Dudziak. Using fictitious play to find pseudo-optimal solutions for full-scale poker. In *2006 International Conference on Artificial Intelligence*, 2006.

[62] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, Nagoya, Japan, 1995.

[63] J.D. Farmer, N. Packard, and A. Perelson. The immune system, adaptation and machine learning. *Physica D*, 2, 1986.

[64] David B. Fogel. *Evolving Artificial Intelligence*. PhD thesis, University of California at San Diego, 1992.

[65] David B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Hoboken, New Jersey, 2006.

[66] David B. Fogel, Lawrence J. Fogel, and V. W. Porto. Evolving neural networks. *Biological Cybernetics*, 63, 1990.

[67] Lawrence J. Fogel. Autonomous automata. *Industrial Research*, 4, 1962.

[68] M. K. Fort. A note on pointwise convergence. *Proc. Amer. Math. Soc.*, 2, 1951.

[69] A. S. Fraser. Simulation of genetic systems by automatic digital computers i: Introduction. *Australian Journal of Biological Science*, 10, 1957.

[70] A. S. Fraser. Simulation of genetic systems by automatic digital computers ii: Effects of linkage on rates of advance under selection. *Australian Journal of Biological Science*, 10, 1957.

[71] R. M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2, 1958.

[72] C. Genovese and L. Wasserman. Rates of convergence for the gaussian mixture sieve. *Annals of Statistics*, 28(4), 2000.

[73] Subhashis Ghosal and Aad W. van der Vaart. Entropies and rates of convergence for maximum likelihood and bayes estimation for mixtures of normal densities. *Annals of Statistics*, 29(5), 2001.

[74] S. Ghosh, S. Das, A.V. Vasilakos, and K. Suresh. On convergence of differential evolution over a class of continuous functions with unique global optimum. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, PP, 2011.

[75] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, Honolulu, Hawaii, 2007.

[76] Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Srensen. A heads-up no-limit texas hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, Estoril, Portugal, 2008.

[77] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[78] David E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4, 1995.

[79] F. Gomez. *Robust Non-linear Control through Neuroevolution*. PhD thesis, 2003.

[80] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research (JMLR)*, 9, 2008.

[81] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5, 1997.

[82] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operation Research*, 13(4), 1988.

[83] P. Halmos. *Measure Theory*. Springer-Verlag, New York, NY, 1974.

[84] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2), 2001.

[85] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317. Morgan Kaufmann, 1996.

[86] G. R. Harik, F. G. Lobo, and D. E. Goldberg. The compact genetic algorithm. In *Proceedings of the International Conference on Evolutionary Computation*, Piscataway, NJ, 1998.

[87] W.K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1), 1970.

[88] S. Haykin. *Neural Networks and Learning Machines, Third Edition.* Prentice Hall, 2008.

[89] D. Heckerman, D. Geiger, and M. Chickering. Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20, 1995.

[90] M. Hestenes and E. Stiefel. *Journal of the Bureau of National Standards*, 49(6), 1952.

[91] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief networks. *Neural Computation*, 18(7), 2006.

[92] Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.

[93] B. Hoehn, F. Southey, R. C. Holte, and V. Bulitko. Effective short-term opponent exploitation in simplified poker. In *Proceedings of the 20th National Conference on Artificial Intelligence*, Madison, WI, 2007. AAAI Press.

[94] J. H. Holland. Outline for a logical theory of adaptive systems. In *Information Processing in the Nervous System, Proceedings of the International Union of Physiological Sciences*, volume 3, 1962.

[95] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, Michigan, 1975.

[96] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *J. ACM*, 8, 1961.

[97] L. Höremainder. *The analysis of linear partial differential operators I.* Springer, 1983.

[98] M. Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of the Foundations of Computer Science*, 2001.

[99] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, Piscataway, NJ, 2003. IEEE Press.

[100] C. Igel and M. Toussaint. A no-free-lunch theorem for non-uniform distributions of target functions. *Journal of Mathematical Modelling and Algorithms*, 3(4), 2004.

[101] T. Jaakkola. *Variational methods for inference and estimation in graphical models.* PhD thesis, Massachussetts Institute of Technology, 1997.

[102] I. Jeong and J. Lee. Adaptive simulated annealing genetic algorithm for system identification. *Engineering Applications of Artificial Intelligence*, 9(5):523 – 532, 1996.

[103] Kenneth A. De Jong, W. M. Spears, and D. F. Gordon. Using markov chains to analyze gafos. *Foundations of Genetic Algorithms 3*, 1995.

[104] M. I. Jordan. Graphical models. *Statistical Science (Special Issue on Bayesian Statistics)*, 19, 2004.

[105] Ioannis Karatzas and Steven Shreve. *Brownian Motion and Stochastic Calculus*. Springer-Verlag, New York, NY, 1991.

[106] William Karush. *Minima of Functions of Several Variables with Inequalities as Side Constraints.* PhD thesis, University of Chicago, Chicago, Illinois, 1939.

[107] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, 1995.

[108] Charles D. Kirkpatrick. *Technical Analysis: The Complete Resource for Financial Market Technicians.*

[109] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), 1983.

[110] Nathan Kohl. *Learning in Fractured Problems for Constructive Neural Network Algorithms.* PhD thesis, University of Texas at Austin, 2009.

[111] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.*, 45, 2003.

[112] A. N. Kolmogorov. *Grundbegriffe der Wahrscheinlichkeitsrechnung.* 1933.

[113] A. N. Kolmogorov. *Foundations of the theory of Probability (English Translation).* Chelsea publishing company, 1956.

[114] Andrey N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1, 1965.

[115] M. Koppen. Some technical remarks on the proof of the no free lunch theorem. In *Proceedings of the Joint Conference on Information Sciences (JCIS 2000)*, 2000.

[116] M. Koppen, D. Wolpert, and M. Macready. Remarks on a recent paper on the "no free lunch" theorems. *IEEE Transactions on Evolutionary Computation*, 5(1), 2001.

[117] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1999.

[118] John R. Koza. *Genetic Programming.* MIT Press, Cambridge, Massachusetts, 1992.

[119] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium*, Berkeley, CA, 1951. University of California Press.

[120] V. Kvasnicka, M. Pelikan, and J. Popischal. Hill climbing with learning (an abstraction of the genetic algorithm). *Neural Network World*, 6, 1996.

[121] Joseph Lagrange. *Méchanique Analytique*. 1788.

[122] Steffen L. Lauritzen. *Graphical Models*. Oxford University Press, New York, 1996.

[123] Henri Lebesgue. *Longueur, Intégrale, Aire*. PhD thesis, Nancy Université, 1902.

[124] Yann LeCun, Koray Kavukvuoglu, and Clémeant Farabet. Convolutional networks and applications in vision. In *Proc. International Symposium on Circuits and Systems (ISCAS'10)*. IEEE, 2010.

[125] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2), 2011.

[126] Lipster and Shiryaev. *Statistics of Random Processes, Second Edition*. Springer-Verlag, New York, New York, 2001.

[127] Kunqi Liu, Xin Du, and Lishan Kang. Differential evolution algorithm based on simulated annealing. In Lishan Kang, Yong Liu, and Sanyou Zeng, editors, *Advances in Computation and Intelligence*, volume 4683 of *Lecture Notes in Computer Science*, pages 120–126. Springer Berlin / Heidelberg, 2007.

[128] A. Lockett, C. Chen, and R. Miikkulainen. Evolving explicit opponent models in game playing. In *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, California, 2007. Kaufmann.

[129] Alan Lockett and Risto Miikkulainen. Evolving opponent models for texas hold 'em. In *2008 IEEE Conference on Computational Intelligence in Games*, December 2008.

[130] Alan Lockett and Risto Miikkulainen. Temporal convolution machines for sequence learning. In *To Appear*, 2009.

[131] Alan Lockett and Risto Miikkulainen. Evolutionary annealing: Global optimization in arbitrary measure spaces. *In Press*, 2011.

[132] Alan Lockett and Risto Miikkulainen. Measure-theoretic evolutionary annealing. In *Proceeedings of the 2011 IEEE Congress on Evolutionary Computation (CEC-2011)*, 2011.

[133] Alan Lockett and Risto Miikkulainen. Real-space evolutionary annealing. In *Proceedings of the 2011 Genetic and Evolutionary Computation Conference (GECCO-2011)*, 2011.

[134] Samir Mahfoud, Samir W. Mahfoud, David E. Goldberg, and David E. Goldberg. Parallel recombinative simulated annealing: A genetic algorithm, 1995.

[135] Paul McQuesten and Risto Miikkulainen. Culling and teaching in neuro-evolution. In *Proc. 7th Intl. Conf. on Genetic Algorithms (ICGA97)*, 1997.

[136] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6), 1953.

[137] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2), 1996.

[138] D. E. Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks.* PhD thesis, University of Texas at Austin, 1997.

[139] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 1991.

[140] Heinz Mühlenbein, T. Mahnig, and A. O. Rodriguez. Schemata, distributions, and graphical models in evolutionary optimization. *Journal of Heuristics*, 5, 1999.

[141] Heinz Mühlenbein and Thilo Mahnig. Mathematical analysis of evolutionary algorithms. In *Essays and Surveys in Metaheuristics, Operations Research/Computer Science Interface Series*, pages 525–556. Kluwer Academic Publisher, 2002.

[142] Heinz Mühlenbein and G Paass. From recombination of genes to the estimation of distributions: Binary parameters. In H. M. Voigt, editor, *Lecture Notes in Computer Science 1141: Parallel Problem Solving from Nature – PPSN IV*, Berlin, 1996. Springer.

[143] Heinz Mülenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm: I. continuous parameter optimization. *Evolutionary Computation*, 1(1), 1993.

[144] James R. Munkres. *Topology.* Prentice Hall, Upper Saddle River, NJ, 2000.

[145] John J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications.*

[146] John Nash. Non-cooperative games. *Annals of Mathmatics*, 54, 1951.

[147] Radford M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[148] Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer, New York, 1996.

[149] J.A. Nelder and R. Mead. A simplex method for function minimization. *Comput. J.*, 7, 1965.

[150] S. Nolfi, J. L. Elman, and D. Parisi. Learning and evolution in neural networks. Technical Report Technical Report 9019, University of California at San Diego, 1990.

[151] Joanna Papakonstantinou. The historical development of the secant method in 1-d. In *The Annual meeting of the Mathematical Association of America*, San Jose, CA, 2007.

[152] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California, 1988.

[153] M.E.H. Pedersen. *Tuning & Simplifying Heuristical Optimization*. PhD thesis, University of Southampton, 2010.

[154] Martin Pelikan, David Goldberg, and Ferdinando Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21, 2002.

[155] Martin Pelikan, David E. Goldberg, and E. Cantu-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999.

[156] Martin Pelikan, David E. Goldberg, and E. Cantu-Paz. Hierarchical problem solving by the bayesian optimization algorithm. Technical Report IlliGAL Report No. 2000002, University of Illinois at Urbana-Champaign, 2000.

[157] Martin Pelikan and Heinz Mülenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing – Engineering Design and Manufacturing*, London, 1999.

[158] D. Plaut, S. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report Technical Report CMU-CS-86-126, Carnegie Mellon University, Pittsburgh, PA, 1986.

[159] M. A. Potter and K. A. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1), 2000.

[160] M.J.D. Powell. Direct search algortihms for optimization calculations. *Acta Numerica*, 1998.

[161] Martin Pring. *Technical Analysis Explained : The Successful Investor's Guide to Spotting Investment Trends and Turning Points.*

[162] Nicholas Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *LECTURE NOTES IN COMPUTER SCIENCE 1000*, pages 275–291. Springer-Verlag, 1995.

[163] Joseph Raphson. *Analysis aequationum universalis.* 1690.

[164] Carl E. Rasmussen. *Evaluation of Gaussian Processes and Other Methods for Non-linear Regression.* PhD thesis, University of Toronto, 1996.

[165] I. Rechenberg. Cybernetic solution path of an experimental problem. *Royal Airport Establishment, Library Translation No. 1122*, 1965.

[166] C. H. Reinsch. Smoothing by spline functions. *Nümerische Mathematik*, 10, 1967.

[167] S. Richardson and P. J. Green. On bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society. Series B (Methodological)*, 59, 1997.

[168] Baker R.J., Cowling P.I., Randall T.W., and Jiang P. Can opponent models aid poker player evolution? In *IEEE Symposium on Computational Intelligence and Games*, Perth, Australia, 2008.

[169] Jani Rönkkönen, Saku Kukkonen, and Jouni Lampinen. A comparison of differential evolution and generalized generation gap model. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, pages 549–555, 2005.

[170] R. Rosenberg. *Simulation of Genetic Populations with Biochemical Properties.* PhD thesis, University of Michigan, Ann Arbor, 1967.

[171] J. E. Rowe, M. D. Vose, and A. H. Wright. Reinterpreting no free lunch. *Evolutionary Computation*, 17(1), 2009.

[172] Jonathan Rubin and Ian Watson. Sartre: System overview. a case-based agent for two-player texas hold'em. In *Eighth International Conference on Case-Based Reasoning*, 2009.

[173] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1), 1994.

[174] A. Ruiz, D. H. Owens, and S. Townley. Existence of limit cycles in recurrent neural networks. In *Industrial Electronics, 1996. ISIE '96., Proceedings of the IEEE International Symposium on*, volume 1, pages 104 –108, 1996.

[175] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. *Computational Models Of Cognition And Perception Series*, 1986.

[176] J. C. Santamaria, R. S. Sutton, and A. Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior*, 6(2), 1998.

[177] Tom Schaul, Yi Sun, Daan Wierstra, Faustino Gomez, and Jürgen Schmidhuber. Curiosity-Driven Optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, 2011.

[178] J. Schmidhuber. The speed prior: A new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT-2002)*, 2002.

[179] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural Computation*, 19(3), 2007.

[180] C. Schumacher, M. D. Vose, and L. D. Whitley. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001*, pages 565–570. Morgan Kaufmann, 2001.

[181] Christopher Wayne Schumacher. *Black box search: framework and methods*. PhD thesis, 2000. AAI9996384.

[182] Gideon E. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2), 1978.

[183] H. P. Schwefel. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stiimungstechnik*. PhD thesis, Technical University of Berlin, 1965.

[184] H. P. Schwefel. *Numerical Optimization of Computer Models*. John Wiley, Chichester, U.K., 1981.

[185] H. T. Siegelmann. *Neural networks and analog computation: beyond the Turing limit*. Birkhauser Boston Inc., Cambridge, MA, USA, 1999.

[186] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York, New York, 1986.

[187] Bryan Silverthorn and Risto Miikkulainen. Latent class models for algorithm portfolio methods. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[188] M. Sipser. *Introduction to the Theory of Computation*. Course Technology PTR, 1996.

[189] David Sklansky and Mason Malmuth. *Hold 'Em Poker for Advanced Players*. 1988.

[190] R. J. Solomonoff. A formal theory of inductive inference. part i. *Information and Control*, 7, 1964.

[191] R. J. Solomonoff. Complexity-based induction systems. *IEEE Transactions on Information Theory*, 24(5), 1978.

[192] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, and D. Billings. Bayes bluff: Opponent modeling in poker. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, 2005.

[193] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4, 1962.

[194] Kenneth O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, 8(2), 2007.

[195] Kenneth O. Stanley and Risto Miikkulainen. Efficient neural network learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, California, 2002. Kaufmann.

[196] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2), 2002.

[197] Kenneth O. Stanley and Risto Miikkulainen. *Efficient Evolution of Neural Networks through Complexification.* PhD thesis, 2004.

[198] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, 1995.

[199] R. Strichartz. *A Guide to Distribution Theory and Fourier Transforms.* CRC Press, 1994.

[200] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambride, MA, 1998.

[201] G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

[202] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1), 1997.

[203] Vinod K. Valsalam, Jonathan Hiller, Robert MacCurdy, Hod Lipson, and Risto Miikkulainen. Constructing controllers for physical multilegged robots using the enso neuroevolution approach. *Evolutionary Intelligence*, 5(1):1–12, 2012.

[204] Massimiliano Vasile, Edmondo Minisci, and Marco Locatelli. An inflationary differential evolution algorithm for space trajectory optimization. *IEEE Transactions on Evolutionary Computation*, 15(2), 2011.

[205] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[206] Michael Vose. *The Simple Genetic Algorithm*. MIT Press, Cambridge, Massachusetts, 1999.

[207] Michael D. Vose. Random heuristic search. *Theoretical Computer Science*, 229:103–142, 1999.

[208] John Wallis. *A Treatise of Algebra both Historical and Practical*. 1685.

[209] C. J. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3), 1992.

[210] Ingo Wegener. On the expected runtime and the success probability of evolutionary algorithms, 2000.

[211] Welling, Rosen-Zvi, and Hinton. Exponential family harmoniums with an application to information retrieval. In *Advances in Neural Information Processing Systems*, 2005.

[212] L. D. Whitley, D. Garrett, and J. P. Watson. Quad search and hybrid genetic algorithms. In *Proceedings of the Genetics and Evolutionary Computation Conference (GECCO-2003)*, volume 2724, Chicago, IL, USA, 2003. Springer.

[213] A. Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, Piscataway, New Jersey, 1991. IEEE Press.

[214] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 18, 2010.

[215] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation (CEC)*, 2008.

[216] P. Wolfe. Convergence conditions for ascent methods. *SIAM Rev.*, 11, 1969.

[217] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[218] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[219] Alden H. Wright. Genetic algorithms for real parameter optimization. In *Foundations of Genetic Algorithms*, pages 205–218. Morgan Kaufmann, 1991.

[220] M.H. Wright. Direct search methods: Once scorned, now respectable. In D.F. Griffiths and G.A. Watson, editors, *Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis*, Harlow, UK, 1996. Addison Wesley Longman.

[221] C.F. Wu. On the convergence properties of the em algorithm. *Annals of Statistics*, 11(1), 1983.

[222] B. Yamauchi and R. Beer. Integration reactive, sequential, and learning behavior using dynamic neural networks. In *From Animals to Animats 3: Proceedings of the 3rd International Conference on SImulation of Adaptive Behavior*. MIT Press, 1994.

[223] R. L. Yang. Convergence of the simulated annealing algorithm for continuous global optimization. *Journal of Optimization Theory and Applications*, 104(3), 2000.

[224] Ming Zhang, Li Yin, and Yongquan Zhou. Hybrid evolution strategies for simultaneous solving all real roots of polynomial. In *International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE-2010)*, 2010.

[225] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems*, volume 20, 2007.

Vita

Alan Justin Lockett was born June 25, 1976. His childhood was spent in North Carolina exploring quartz-filled ravines around his family home and careening down hills on his bike, occasionally on the wrong side of the road, with predictable consequences. He moved to Austin, Texas on June 26, 1992, one day after failing a driving test, consigning him to beg rides from friends for two more years. In Austin, he studied violin at the University of Texas with Prof. Eugene Gratovitch and composed string quartets and a symphony that have yet to be performed by actual musicians. In 1994, he enrolled in the University of Texas, where he studied Linguistics, Classics, and Greek, obtaining the Bachelor of Arts in 1998 with a minor in Mathematics. Afterwards, he traveled briefly on five of the seven continents, and upon settling down, taught Latin for two years in the upper school of Regents School of Austin. He then returned to the University of Texas for a Master of Arts in Middle Eastern Studies, which he completed in 2003 with a report on contemporary Turkish politics supervised by Prof. Clement Henry. During that time, he gave himself multiple scars on a mountain bike. He also studied Computer Science, and subsequently worked as a Research Engineer at 21st Century Technologies Inc., where, under the mentoring of Dr. Paula de Witte and Prof. Victor Raskin, he won a $\$500,000$ Phase II SBIR grant in 2004, on which he eventually served as Principal Investigator. In 2006, he once again entered the graduate program at University of Texas at Austin in 2006, this time in the Department of Computer Science. He obtained a Master of Science in Computer Science in 2007 after studying how autonomous computer poker players could develop computational models of their opponents in order to improve their play. He has since studied under Prof. Risto Miikkulainen, researching topics such as the estimation of temporal probabilistic models, the theory of evolutionary computation, and learning neural network controllers.

Permanent address: 4701 Monterey Oaks Blvd Apt 227
Austin, Texas 78749

This dissertation was typeset with LaTeX[†] by the author.

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.