The Dissertation Committee for David Merrill Pardoe
certifies that this is the approved version of the following dissertation:

Adaptive Trading Agent Strategies Using Market Experience

Committee:

Peter Stone, Supervisor

Risto Miikkulainen

Raymond Mooney

Maytal Saar-Tsechansky

Michael Wellman

Adaptive Trading Agent Strategies Using Market Experience

by

David Merrill Pardoe, B.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

August 2011

Acknowledgments

Thanks to my parents for the Commodore 64 (among many other things), to my advisor for getting me interested in this line of work and supporting me along the way, and to all of the other researchers I interacted with, especially the organizers of and participants in the Trading Agent Competition.

Adaptive Trading Agent Strategies Using Market Experience

Publication No. ____________________

David Merrill Pardoe, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Peter Stone

Along with the growth of electronic commerce has come an interest in developing autonomous trading agents. Often, such agents must interact directly with other market participants, and so the behavior of these participants must be taken into account when designing agent strategies. One common approach is to build a model of the market, but this approach requires the use of historical market data, which may not always be available. This dissertation addresses such a case: that of an agent entering a new market in which it has no previous experience. While the agent could adapt by learning about the behavior of other market participants, it would need to do so in an online fashion. The agent would not necessarily have to learn from scratch, however. If the agent had previous experience in similar markets, it could use this experience to tailor its learning approach to its particular situation.

This dissertation explores methods that a trading agent could use to take advantage of previous market experience when adapting to a new market. Two distinct learning settings are considered. In the first, an agent acting as an auctioneer must adapt the parameters of an auction mechanism in response to bidder behavior, and a reinforcement learning approach is used. The second setting concerns agents that must adapt to the behavior of competitors in two scenarios from the Trading Agent Competition: supply chain management and ad auctions. Here, the agents use supervised learning to model the market. In both settings, methods of adaptation can be divided into four general categories: i) identifying the most similar previously encountered market, ii) learning from the current market only, iii) learning from the current market but using previous experience to tune the learning algorithm, and iv) learning from both the current and previous markets. The first contribution of this dissertation is the introduction and experimental validation of a number of novel algorithms for market adaptation fitting these categories. The second contribution is an exploration of the degree to which the quantity and nature of market experience impact the relative performance of methods from these categories.

Table of Contents

Acknowledgments **iv**

Abstract **v**

List of Tables **xiv**

List of Figures **xvi**

Chapter 1. Introduction **1**

1.1 Categorization of Adaptive Approaches 4
1.2 Contributions 7
1.3 Organization 8

Chapter 2. Adaptive Auction Mechanisms **12**

2.1 Introduction 12
2.2 Adaptive Auction Mechanisms 15
2.2.1 Learning Problem 18
2.2.2 Bandit Adaptive Algorithm 19
2.2.3 Regression Adaptive Algorithm 20
2.3 Auction Settings 22
2.3.1 Reserve Price Setting 23
2.3.2 BIN Price Setting 27
2.4 Experiments 31
2.4.1 Reserve Price for Loss Averse Bidders 32
2.4.2 BIN Price for Time Sensitive Bidders 33
2.5 Summary 34

Chapter 3. Adaptive Auction Mechanisms with Prior Knowledge **36**

3.1 Prior Knowledge of Bidder Behavior 36
3.2 Algorithms 38
3.2.1 Metalearning 38

3.2.2 Bayesian Approach 41
3.3 Experiments 44
3.3.1 Reserve Price Setting 44
3.3.1.1 Correct Prior Knowledge 44
3.3.1.2 Incorrect Prior Knowledge - Random Bidders 47
3.3.1.3 Incorrect Prior Knowledge - Parameter Ranges . . . 48
3.3.1.4 Non-stationary Bidders 50
3.3.2 BIN Price Setting 53
3.3.2.1 Correct Prior Knowledge 54
3.3.2.2 Incorrect Prior Knowledge - Exponential Discounting 56
3.3.2.3 Best-response Bidders 58
3.4 Summary 59

Chapter 4. The TacTex Agent for Supply Chain Management 61
4.1 TAC SCM Overview 61
4.2 TAC SCM Specification 62
4.2.1 Component Procurement 63
4.2.2 Computer Sales 65
4.2.3 Production and Delivery 65
4.3 Overview of TacTex 66
4.3.1 Agent Components 66
4.4 The Demand Manager 70
4.4.1 Demand Model 70
4.4.2 Offer Acceptance Predictor 71
4.4.3 Demand Manager 72
4.4.3.1 Production Scheduling Algorithm 73
4.4.3.2 Handling Existing Orders 74
4.4.3.3 Bidding on RFQs and Handling Predicted Orders . . 74
4.4.3.4 Completing Production and Delivery 78
4.4.3.5 Production Beyond 10 Days 78
4.5 The Supply Manager 79
4.5.1 Supplier Model 79
4.5.1.1 Price Prediction 79
4.5.1.2 Reputation 81

4.5.1.3 Price Probes 81
4.5.2 Supply Manager 81
4.5.2.1 Deciding What to Order 82
4.5.2.2 Deciding How to Order 82
4.5.2.3 2-Day RFQs 84
4.6 2005 Competition Results 85
4.7 Experiments 86
4.7.1 The Three Predictor Modules 88
4.7.1.1 Offer Acceptance Predictor 89
4.7.1.2 Demand Model 90
4.7.1.3 Supplier Model 90
4.7.2 Experiments with the Supply Manager 91
4.7.2.1 2-day RFQs 91
4.7.2.2 Inventory Threshold 91
4.7.2.3 Reduced RFQ Flexibility 93
4.7.2.4 Range of Days Considered 93
4.7.2.5 The Effect of Opponent Strategies 95
4.8 Summary 95

Chapter 5. Learning for Price Prediction in TAC SCM 96
5.1 Learning for Computer Price Prediction 96
5.1.1 Particle Filter for Price Distribution Estimation 96
5.1.2 Future Price Changes 98
5.1.3 2006 Competition Results 101
5.1.4 Computer Price Prediction Experiments 103
5.2 Learning for Component Price Prediction 105
5.3 The SCM Prediction Challenge 107
5.3.1 Challenge Specification 108
5.3.2 Prediction Methods 112
5.3.3 2007 Results 114
5.3.4 Average Daily Errors 115
5.3.5 Differences Between Participants Across Games 120
5.3.6 Differences Between Participants Across Days 125
5.3.7 Error Persistence 129

5.3.8 Individual Error Distribution . . . 131
5.3.9 2008 Results . . . 132
5.4 Learning Across Markets . . . 133
5.4.1 Generalization . . . 135
5.4.2 Learning Online . . . 141
5.5 Summary . . . 143

Chapter 6. The TacTex Agent for Ad Auctions 146
6.1 TAC AA Description . . . 146
6.1.1 Overview . . . 147
6.1.2 Users . . . 148
6.1.3 Click model . . . 148
6.1.4 Auctions . . . 149
6.1.5 Capacity . . . 149
6.1.6 Information . . . 150
6.1.7 Example . . . 150
6.2 TacTex Overview . . . 151
6.3 Position Analyzer . . . 153
6.4 User Model . . . 153
6.5 Advertiser Model . . . 155
6.5.1 Impression Predictions . . . 157
6.5.2 Ad predictions . . . 157
6.5.3 Advertiser Bid Estimation . . . 157
6.5.3.1 First Bid Estimator . . . 158
6.5.3.2 Second Bid Estimator . . . 160
6.6 Parameter Model . . . 164
6.7 Optimization . . . 165
6.7.1 Query Analyzer . . . 166
6.7.2 Single-day Optimizer . . . 168
6.7.3 Multi-day Optimizer . . . 170
6.7.4 First Two Days . . . 172
6.8 2009 Agent Performance . . . 173
6.8.1 2009 Competition . . . 173
6.8.2 Experiments . . . 175
6.9 2010 Results . . . 179
6.10 Summary . . . 180

Chapter 7. Learning for Bid Estimation in TAC AA **181**
7.1 An Improved Bid Estimator 181
7.1.1 Auction Setting 183
7.1.2 Particle Filtering 184
7.1.2.1 SIS Particle Filter 185
7.1.2.2 Choice of Proposal Distribution 186
7.1.2.3 Computing Pr(report | p) 187
7.1.2.4 Sampling from Pr(p′ | p, report) 190
7.1.2.5 Example 190
7.1.2.6 Resampling 191
7.1.3 Application to TAC AA 191
7.1.4 Advertiser Models 193
7.1.5 Experiments 196
7.1.5.1 Setup 197
7.1.6 Estimation Results 199
7.1.7 Application to Bidding 201
7.2 Learning Advertiser Models 202
7.2.1 Generalization 204
7.2.2 Learning Online 206
7.2.3 Estimation Accuracy 208
7.3 Summary 210

Chapter 8. Transfer Learning for Regression **211**
8.1 Problem Specification 211
8.2 Algorithms 212
8.2.1 Base Learner on Target Data 213
8.2.2 Best Source Model 214
8.2.3 Best Source Weighting 214
8.2.4 Transfer Stacking 215
8.2.5 Feature Augmentation 216
8.2.6 Boosting Approaches 218
8.2.6.1 AdaBoost and Regression 219
8.2.6.2 ExpBoost.R2 222
8.2.6.3 TrAdaBoost.R2 222

8.2.6.4 Two-stage TrAdaBoost.R2 224
8.2.7 TrBagging 226
8.2.8 Discussion of Algorithms 228
8.3 Data Transformation 232
8.4 Experiments 233
8.4.1 Four UCI Data Sets 234
8.4.2 Friedman #1 238
8.4.3 TAC Data Sets 241
8.4.4 Additional Experiments with Two-Stage TrAdaBoost.R2 . . . 246
8.5 Summary 250

Chapter 9. Using Past Game Experience in TAC **251**
9.1 Single Sources of Varying Similarity 252
9.2 Source and Target are Identical 254
9.3 Multiple Sources 260
9.4 Source Weighting in `2ST` 264
9.5 Limited Source Data 266
9.6 Learning from TAC Games 270
9.7 SCM Agent Performance 271
9.8 Transfer in a Non-stationary Setting 274
9.9 Summary 276

Chapter 10. Related Work **277**
10.1 Adaptive Auction Mechanisms 277
10.2 Supply Chain Management 283
10.2.1 Price Prediction 284
10.2.2 Bidding on Customer RFQs 286
10.2.3 Production Scheduling 286
10.2.4 Component Procurement 287
10.3 Ad Auctions 287
10.4 Regression Transfer 290
10.5 Summary 292

Chapter 11. Conclusion **294**
11.1 Summary of Contributions 294
11.2 Future Work 297
11.2.1 Sequential Auctions 297
11.2.2 TAC Agents 300
11.2.3 Regression Transfer 302
11.3 Conclusion 304

Appendix **305**

Appendix 1. Position Analyzer for Ad Auctions **306**

Bibliography **315**

Vita **329**

List of Tables

1.1 Market scenarios considered 10
1.2 Contributions in each chapter 11
1.3 Stand-alone topics 11

2.1 Average revenue per auction for each adaptive algorithm 33

3.1 Parameters found through metalearning. 41
3.2 Average revenue per auction for each adaptive algorithm 46
3.3 Average revenue when parameters drawn from modified distributions 50
3.4 Average revenues, non-stationary population 53
3.5 Average profit for BIN price setting 55
3.6 Average profit for exponential discounters 57
3.7 Average profit for best-response bidders 59

4.1 Overview of the steps taken each day by TacTex. 69
4.2 Results of the 2005 final round 86
4.3 Experimental results - 2005 agent 88

5.1 Features for learning changes in computer prices. 100
5.2 Results of the 2006 final round 101
5.3 Experimental results - 2006 agent 104
5.4 Features for learning changes in component prices. 106
5.5 Experimental results for the component price change predictor. . . 106
5.6 PC07 Current computer prices 114
5.7 PC07 Future computer prices 114
5.8 PC07 Current component prices 114
5.9 PC07 Future component prices 114
5.10 PC07 Overall placing and average rank of each participant 115
5.11 Agents in each of the three sets of games 115
5.12 RMS errors over different periods for each category 120
5.13 PC08 Current computer prices 134

5.14 PC08 Future computer prices 134
5.15 PC08 Current component prices 134
5.16 PC08 Future component prices 134
5.17 PC08 Overall placing and average rank of each participant 134
5.18 Agents in each of the three sets of games, 2008 134
5.19 Model accuracy across groups - computer costs 137
5.20 Model accuracy across groups - component costs 138
5.21 Impact of model choice on agent performance. 141

6.1 Results for the query *null:dvd* from one game day 151
6.2 TAC AA Experimental results 176

7.1 Bid estimate errors for all estimators - setting 1 201
7.2 Bid estimate errors for all estimators - setting 2 201
7.3 Bid estimate errors for all estimators - setting 3 201
7.4 Model accuracy between agents 205
7.5 Bid estimate errors using different models - setting 2 (QuakTAC7) . 209
7.6 Bid estimate errors using different models - setting 3 (AstonTAC7) 209

8.1 Properties of the regression transfer algorithms 230
8.2 RMS error on four UCI datasets 237
8.3 Summary of results in Table 8.2 237

9.1 Agent scores with transfer 273

1.1 Results for the query *null:dvd* from one game day 307
1.2 Values computed by the Position Analyzer 308

List of Figures

2.1 A high-level illustration of the concept of adaptive mechanisms. . . 17
2.2 Results for different loss averse bidder populations 27
2.3 Results for different time-sensitive bidder populations 31
2.4 Average revenue per auction over the course of an episode - reserve 34
2.5 Average revenue per auction over the course of an episode - BIN . . 35

3.1 Results for different reserve prices 40
3.2 Metalearned parameters 45
3.3 Average revenue per auction over the course of an episode - reserve 47
3.4 Results when some fraction of bidders choose a bid randomly. . . . 49
3.5 Average revenue per auction, different distribution 50
3.6 Metalearned parameters 55
3.7 Average profit per auction over the course of an episode - BIN . . . 55
3.8 Number of times each BIN price was optimal 56
3.9 Average profit, exponential discounters 57
3.10 Average profit over episode, learning bidders 60

4.1 The TAC SCM Scenario 63
4.2 An overview of the main agent components 68

5.1 Average computer prices 99
5.2 Daily profits for the top three agents 102
5.3 Current computer prices (avg. over all games) 117
5.4 Future computer prices (avg. over all games) 117
5.5 Current component prices (avg. over all games) 117
5.6 Future component prices (avg. over all games) 117
5.7 Current computer prices (game A-3) 119
5.8 Future computer prices (game A-3) 119
5.9 Current component prices (game A-3) 119
5.10 Future component prices (game A-3) 119

5.11 Current computer prices (each game, TT / DM) 121
5.12 Future computer prices (each game, TT / DM) 121
5.13 Current component prices (each game, TT / DM) 121
5.14 Future component prices (each game, TT / DM) 121
5.15 Current computer prices (each game, Bot. / DM) 122
5.16 Future computer prices (each game, Bot. / DM) 122
5.17 Current component prices (each game, Bot. / DM) 122
5.18 Future component prices (each game, Bot. / DM) 122
5.19 Current computer prices (each game, TT / Bot.) 123
5.20 Future computer prices (each game, TT / Bot.) 123
5.21 Current component prices (each game, TT / Bot.) 123
5.22 Future component prices (each game, TT / Bot.) 123
5.23 Current computer prices (each day, TT / DM) 126
5.24 Future computer prices (each day, TT / DM) 126
5.25 Current computer prices (each day, TT / Bot.) 127
5.26 Future computer prices (each day, Bot. / DM) 127
5.27 Current component prices (each day, TT / DM) 127
5.28 Future component prices (each day, TT / DM) 127
5.29 Current computer prices (daily differences with TT) 128
5.30 Future computer prices (daily differences with TT) 128
5.31 Current component prices (daily differences with DM) 128
5.32 Future component prices (daily differences with TT) 128
5.33 Current component prices (errors on consecutive days, TT) 130
5.34 Current computer prices (daily differences, Bot-TT) 131
5.35 Current component prices (daily differences, DM-TT) 131
5.36 Current computer price error distribution for TacTex 133
5.37 Future computer price error distribution for TacTex 133
5.38 Current component price error distribution for TacTex 133
5.39 Future component price error distribution for TacTex 133
5.40 Dendrogram of computer cost similarity based on Table 5.19 139
5.41 Dendrogram of component cost similarity based on Table 5.20 . . . 140
5.42 Computer price prediction error for TT10 when learned online . . . 144
5.43 Component price prediction error for TT10 when learned online . . 144
5.44 Computer price prediction error for DM09 when learned online . . . 145

5.45 Component price prediction error for DM09 when learned online . . 145

6.1 Flow of information in TacTex 152
6.2 User population estimates for one product 156
6.3 User population estimates for a second product 156
6.4 2009 final round scores (in thousands) 174
6.5 2010 final round scores (in thousands) 180

7.1 Daily bids of two advertisers 198
7.2 Top five daily bids 199
7.3 Dendrogram of bid model similarity based on Table 7.4 206
7.4 Bid model error for QuakTAC when learned online 207
7.5 Bid model error for AstonTAC when learned online 207
7.6 Bid model error for Schlemazl1 when learned online 208

8.1 Illustration of Friedman #1 239
8.2 Friedman #1 with 1 source 239
8.3 Friedman #1 with 5 sources 240
8.4 Friedman #1 with 1 and 5 sources 241
8.5 TAC Travel 243
8.6 TAC SCM computers 244
8.7 TAC SCM components 244
8.8 TAC AA bid probability 245
8.9 Two-stage alternatives, TAC Travel 247
8.10 Two-stage alternatives, TAC SCM computers 248
8.11 Two-stage alternatives, TAC SCM components 248
8.12 Source weight, TAC Travel 249
8.13 Source weight, TAC SCM computers 249
8.14 Source weight, TAC SCM components 249

9.1 One poor source, SCM-CPU 253
9.2 One poor source, SCM-CPO 253
9.3 One poor source, AA 253
9.4 One good source, SCM-CPU 255
9.5 One good source, SCM-CPO 255
9.6 One good source, AA 255

9.7 One medium source, SCM-CPU . . . 256
9.8 One medium source, SCM-CPO . . . 256
9.9 One medium source, AA . . . 256
9.10 `2ST` with each source, SCM-CPU . . . 257
9.11 `2ST` with each source, SCM-CPO . . . 257
9.12 `BSW` with each source, AA . . . 257
9.13 Source equals target, SCM-CPU . . . 259
9.14 Source equals target, SCM-CPO . . . 259
9.15 Source equals target, AA . . . 259
9.16 Add poor source to good source, SCM-CPU . . . 262
9.17 Add poor source to good source, SCM-CPO . . . 262
9.18 Add poor source to good source, AA . . . 262
9.19 Add medium to good source, SCM-CPU . . . 263
9.20 Add medium to good source, SCM-CPO . . . 263
9.21 Add medium to good source, AA . . . 263
9.22 All 3 sources, SCM-CPU . . . 265
9.23 All 3 sources, SCM-CPO . . . 265
9.24 All 3 sources, AA . . . 265
9.25 Uniform source weights in first stage, SCM-CPO . . . 266
9.26 Limited good source, SCM-CPU . . . 268
9.27 Limited good source, SCM-CPO . . . 268
9.28 Limited good source, AA . . . 268
9.29 Limited good and full medium source, SCM-CPU . . . 269
9.30 Limited good and full medium source, SCM-CPO . . . 269
9.31 Limited good and full medium source, AA . . . 269
9.32 Learning from TAC games, SCM-CPU . . . 272
9.33 Learning from TAC games, SCM-CPO . . . 272
9.34 Learning from TAC games, AA . . . 272
9.35 Non-stationary setting, SCM-CPU . . . 275
9.36 Non-stationary setting, SCM-CPO . . . 275

1.1 Part of the search tree for Table 1.1 . . . 312

Chapter 1

Introduction

The growth of electronic commerce in recent years has been extremely rapid. For example, North American online retail sales were expected to reach $329 billion in 2010, up from $39 billion in 2000[1], and business-to-business sales are estimated at many times this amount. One of the most impressive successes of e-commerce has been the online auction site eBay, which recorded $57.2 billion in sales from 2.9 billion item listings in 2009[2]. While most e-commerce transactions currently involve at least one human participant, autonomous trading agents are likely to represent an increasingly large portion of e-commerce participants in the future, and the development of such agents has become an important area of research.

In some cases, agents are able to operate independently of other market participants. For example, a "price bot" (an agent that searches the Internet for the lowest posted price for a given item) may not need to take into account the actions of other shoppers. In many cases, however, an agent must interact directly with other participants in the market and may need to reason about these participants. For instance, an agent bidding in an auction should take into account the possible actions of other bidders, while an agent managing an online store should consider the preferences of customers and the prices of competitors. Thus, the development of trading agents generally falls under the subfield of Artificial Intelligence (AI) known as Multiagent Systems (MAS).

[1] http://en.wikipedia.org/wiki/Business-to-consumer
[2] http://www.annualreports.com/HostedData/AnnualReports/PDFArchive/ebay2009.pdf

One common approach to designing strategies for trading agents is an empirical one. When sufficient historical market data is available, it may be possible to learn a model that captures the behavior of market participants and allows the optimal strategy with respect to this behavior to be determined. For instance, data mining is often applied to the extensive database of completed eBay auctions to determine the auction settings that will maximize seller revenue for a given item [92].

An alternative approach to designing agent strategies, both for market applications in particular and within MAS in general, is through the use of game theory. Given assumptions about agent preferences over various outcomes (which may themselves be based on empirical data), an agent or agent designer could identify a strategy profile (a set of strategies for all participating agents) that represents an equilibrium, i.e., one from which no agent has an incentive to unilaterally deviate. Behaving according to the strategy specified for the agent would thus be optimal in expectation, so long as all other agents do the same. In some cases, however, there may be multiple equilibria, leaving an agent unsure of which strategy to choose. Also, identifying an equilibrium may be intractable in sufficiently complex domains. Even when a unique equilibrium can be found, there is no guarantee that all agents will behave as specified by the equilibrium. Agents may deviate from their expected behavior for many reasons, including incorrect assumptions about the preferences of other agents and simple irrationality. Experiments in the field of experimental economics frequently reveal what appears to be irrational behavior in many scenarios, including market settings (e.g., [50]).

The problem addressed in this dissertation, however, is that of an agent entering a new market in which it has no previous experience and for which no past data is available. (In this dissertation, a reference to a specific market indicates a particular group of interacting buyers and sellers of some specific product or category of products.) This situation might arise if the agent is selling a new product that does not directly compare to any existing product, for example. An agent in this situation might choose not to use the game-theoretic approach (perhaps due to lack of assumptions about the preferences of other market participants) and would be unable to use the empirical approach described above (due to the lack of existing data). The agent could still take a learning approach, but the learning would need to come in the form of adapting to the behavior of other market participants in an online fashion.

Existing implementations of agents that can adapt in a new market typically involve learning from scratch. If the agent truly has no previous experience or knowledge of any kind, then there may be no alternative. In some cases, however, an agent may have experience in different markets prior to its entry into the new market. In other cases, the agent may have an idea of what types of behavior might be exhibited by other market participants, raising the possibility of generating simulated experience. In either situation, the agent should be able to use the additional information to tailor the learning algorithm to the situation and do better than learning from scratch. This dissertation will answer the question of how this information can be used:

How can adaptive trading agents take advantage of previous experience (real or simulated) in other markets, while remaining robust in the face of novel situations in a new market?

Within the context of a specific market scenario, the answer to this question depends on the following four questions:

1. What does the agent need to learn about to adapt?
2. What sources of previous market experience are available? More specifically:
    - If using existing data from other markets, which markets are relevant?

    - If using simulation, what behaviors will be used by the simulated market participants?

3. What form of learning will be used to adapt?

4. How will this previous market experience be used to improve the adaptive approach?

The answers to the first three questions depend on the particular market scenario being considered. This dissertation will cover three distinct scenarios for which the answers to these questions differ. In particular, two distinct forms of learning will be considered:

- *Reinforcement Learning:* By observing the results of its own actions, the agent directly learns a policy that maps states to actions [102].

- *Supervised Learning:* The agent learns a model of the market from its experience. The outputs of this model are then used as inputs to an optimization routine that chooses actions. The agent may choose to model the aggregate effect of the other market participants or to model each market participant individually.

Additional details on the three market scenarios will be presented in Section 1.3. The goal of this dissertation will be to provide answers to the fourth question by presenting methods that can be used to improve the performance of specific learning algorithms in a domain-independent way. The following section discusses these approaches at a high level.

1.1 Categorization of Adaptive Approaches

The methods that an agent could use to adapt to a new market can be divided into four basic approaches (which are not necessarily mutually exclusive and may be combined in some cases):

A. Learn from the new market only.

B. Attempt to identify the market, or maintain a distribution over possible markets, from a space of possible markets. In this case, information about previous or simulated markets helps us choose this space, while information from the new market is used to identify the market.

C. Choose a learning method that considers only information from the new market, but is tuned through experience in previous markets. This tuning could take a variety of forms, such as choosing initial parameters or the structure of a model.

D. Learn from both the new market and the previous markets.

D.1. Learn using a combination of data from all sources.

D.2. Learn using a combination of models from all sources.

The first can be described as "combine the data, then model it", while the second can be described as "model the data, then combine the models". In both cases, the challenge is to identify which data/models are relevant, and then determine how to perform the combination.

The effectiveness of each approach depends on the attributes of the problem to which it is applied. Among the attributes of market adaptation problems that will be considered in this dissertation, and their possible impacts on the effectiveness of the adaptive approaches, are:

1. *Whether the problem presents an exploration-exploitation tradeoff.* If the agent can choose its own training examples through its actions but is rewarded based on the actions chosen, then the tradeoff between exploration

and exploitation needs to be addressed. Different approaches to using previous experience may offer different clues about which actions offer both high reward and useful information. Approach B allows the possibility of taking actions that help differentiate between possible markets, while approach C could involve choosing a suitable exploration method.

2. *How similar the previous and new markets are.* The greater the differences, the less useful past experience will be, and the more likely it is that approach A is best. If the new market is greatly dissimilar to the previous markets, approach B may fail. In approach C, the fact that the learning method is biased for a particular set of markets may slow initial learning, but it may still be possible to learn effectively.

3. *The amount of experience available (in both the previous markets and the new market).* Once sufficient experience in the new market is available, it is unlikely that any approach could do better than approach A, but different approaches might be effective at different points before this. If only a little experience is available in a previous market, approach D.1 might be more effective than approach D.2, since the models in approach D.2 might have low accuracy. Limited experience in a previous market might also reduce the ability to either accurately identify the market (approach B) or effectively tune the learning approach (C). Another way that experience in previous markets can be measured is by the *number* of previous markets from which this experience comes. Increasing the number of markets could improve the performance of approaches B, C, and D, but could also introduce additional challenges.

4. *Whether (or to what degree) the new market is nonstationary.* If the market is nonstationary, then in a sense this is similar to perpetually having little

experience in the current state of the new market, making previous experience more valuable. If the same type of nonstationarity is also present in the previous markets, then it may be possible to identify the nature of the nonstationarity (approach B), or tune the learning method to the type of nonstationarity expected (approach C).

One contribution of this dissertation will be experimenting to confirm the degree to which these problem attributes impact the effectiveness of the different adaptive approaches.

1.2 Contributions

In this dissertation, I will present adaptive trading agents in three distinct market scenarios. The specific contributions of the dissertation within each scenario can be divided as follows:

Contribution 1 A complete specification of an autonomous trading agent designed for the scenario, with experimental verification that the agent performs well.

Contribution 2 Learning methods the agent can use to adapt to other agents, and experimental verification that this adaptation is beneficial.

Contribution 3 Multiple approaches for making use of previous market experience, including approaches from the categories described above.

Contribution 4 Experiments showing the effectiveness of the adaptive approaches under different conditions, including those described above, along with conclusions about when each approach is best.

To summarize the results that will be presented as part of Contribution 4, for each of two different learning settings (reinforcement learning and supervised regression), one approach will be identified that performs particularly well across a variety of conditions. In the reinforcement learning setting, this approach will be a metalearning approach to setting learning parameters of a particular reinforcement learning algorithm, fitting category C above. In the supervised regression setting, this approach will be a boosting algorithm (2-Stage TrAdaBoost.R2) that can effectively make use of training data from multiple sources, fitting category D.1 above. These learning approaches are generally applicable (i.e., not specific to market domains), and their introduction and experimental verification represents the central achievement of this dissertation.

1.3 Organization

The first market scenario I will discuss is one in which an auctioneer running multiple auctions in one market (such as an eBay seller with multiple identical items) must adapt an auction mechanism in response to bidder behavior in order to maximize some objective function such as total revenue. Here the auctioneer faces an active learning problem and must handle the tradeoff between exploring different mechanisms and exploiting the mechanism currently believed to be best. For this reason RL methods are likely to be effective, although it is also possible to attempt to directly model bidders. In this scenario, it is assumed that the auctioneer has some idea of behaviors that are likely to be exhibited by bidders and can therefore simulate a range of plausible bidder populations, and these simulations represent the available previous market experience.

The next two market scenarios come from the Trading Agent Competition (TAC), a series of annual competitions that has covered several different market scenarios. One barrier to trading agent research is that it can be difficult to benchmark automated strategies in a live business environment due to both the proprietary nature of the systems and the high cost of errors. TAC provides testbeds for studying and prototyping agents by providing a competitive environment in which independently created agents can be tested against each other over the course of many simulations in an open academic setting. A particularly appealing feature of TAC is that, unlike in many simulation environments, the other agents are externally-developed profit-maximizing agents with incentive to perform well, rather than strawman benchmarks. In addition, after each year's competition, binaries of most competing agents are released, making it possible to perform controlled experiments after the fact. Within each of the two TAC scenarios considered in this dissertation I will present a winning agent and then show how making use of previous market experience can further improve the performance of this agent.

The first TAC scenario I discuss is the Supply Chain Management scenario (TAC SCM). In TAC SCM, six agents compete as computer manufacturers and must purchase computer components, assemble computers, and sell the computers to customers. Performing these tasks effectively requires making predictions about quantities such as future prices, and these quantities can depend heavily on the competing agents. Thus for the TAC SCM scenario, adaptation takes the form of learning predictors for a specific market (i.e., combination of opponents) through the use of supervised learning methods for regression problems. Previous market experience comes in the form of games played against different sets of agents.

The second TAC scenario I discuss is the Ad Auctions scenario (TAC AA). In TAC AA, eight agents compete as advertisers to see who can make the most profit from selling a limited range of home entertainment products. Agents negotiate with a search engine to have their ads shown alongside search results by bidding in an auction each time a search is performed (known as a keyword auction). As in TAC SCM, an effective agent must be able to make predictions about a number of factors. In TAC AA, adaptation takes the form of learning models of the bidding patterns

Market Scenario	sell in sequential auctions	TAC SCM	TAC AA
Learning Setting	reinforcement learning	supervised learning (regression)	
What to Learn	auction parameters	product prices	bidding models
Who Learned for	bidder population	set of opponents	single agent
Previous Experience	simulated bidders	sets of games in different markets	

Table 1.1: Market scenarios considered

of each individual opponent. Again, supervised learning methods for regression problems are used, and previous market experience comes in the form of games against different opponents.

Table 1.1 shows a summary of the learning approaches taken for the three different market scenarios considered in this dissertation.

The remainder of this dissertation is organized as follows. I first present work on the sequential auction scenario. Chapter 2 describes the scenario in detail and presents an agent that can learn auction parameters in response to bidder behavior. Chapter 3 discusses improvements to the agent that allow it to make use of prior experience in the form of simulated bidders. I then describe the two TAC scenarios. Chapter 4 introduces TAC SCM and presents our winning agent. Chapter 5 then goes into detail on how the agent can use machine learning to make predictions about computer and component prices. Chapter 6 describes TAC AA and presents our winning agent. Chapter 7 then discusses a method by which an agent can estimate the bids of other agents, including the use of machine learning to model bidding patterns. Next, I present methods of using previous market experience that can be used by both TAC agents. Chapter 8 addresses the general problem of transfer learning for regression problems and introduces several algorithms for this problem. Chapter 9 then further explores the use of these algorithms within the TAC scenarios. Table 1.2 maps out how the content of Chapters 2 through 9 satisfies the contributions promised in the previous section. Alternatively, Table 1.3 lists a division of these chapters into four topics (adaptive auction

	sell in sequential auctions	*TAC SCM*	*TAC AA*
Chapter 2	Contribution 1 + 2		
Chapter 3	Contribution 3 + 4		
Chapter 4		Contribution 1	
Chapter 5		Contribution 2	
Chapter 6			Contribution 1
Chapter 7			Contribution 2
Chapter 8		Contribution 3	
Chapter 9		Contribution 4	

Table 1.2: Contributions in each chapter

Topic	*Chapters to Read*
Adaptive Auction Mechanisms	Chapters 2 and 3
The TacTex Agent for TAC SCM	Chapters 4 and 5
The TacTex Agent for TAC AA	Chapters 6 and 7
Transfer Learning for Regression	Chapters 8 and 9 (Reading Chapters 5 and 7 will give background on the data sets used, but is not required.)

Table 1.3: Stand-alone topics

mechanisms, the TacTex agent for TAC SCM, the TacTex agent for TAC AA, and transfer learning for regression) that can be read about independently. Chapter 10 discusses related work, including both work on the specific e-commerce settings studied in this dissertation and work related to the general learning methods used for agent adaptation. Chapter 11 concludes with a summary of the dissertation and a discussion of areas for future work.

Chapter 2

Adaptive Auction Mechanisms

Auction mechanism design has traditionally been a largely analytic process, relying on assumptions such as fully rational bidders. In practice, however, bidders often exhibit unknown and variable behavior, making them difficult to model and complicating the design process. To address this challenge, in the next two chapters I explore the use of an adaptive auction mechanism: one that *learns* to adjust its parameters in response to past empirical bidder behavior so as to maximize an objective function such as the seller's revenue. In this chapter, I give an overview of the general approach and then present instantiations in two specific sequential auction settings. Results indicate that the adaptive mechanism has the potential to outperform any single fixed mechanism.

An adaptive auction mechanism can be viewed as an autonomous agent designed to trade through selling in sequential auctions, and as such the learning approach presented in this chapter fulfills Contributions 1 and 2 for the sequential auction scenario, as outlined in Section 1.2. In the next chapter, I will show how prior knowledge of possible bidder behavior can be incorporated into the adaptive mechanism to substantially improve performance under varying circumstances, fulfilling Contributions 3 and 4.

2.1 Introduction

Recent years have seen the emergence of numerous auction platforms that cater to a variety of markets such as business to business procurement and consumer to consumer transactions. Many different types of auction *mechanisms* defining the rules of exchange may be used for such purposes. Varying parameters of the auction mechanism, such as auctioneer fees, minimum bid increments, and reserve prices, can lead to widely differing results depending on factors such as bidder strategies and product types. This chapter addresses the possibility of *learning* auction parameters so as to maximize an objective function, such as auctioneer revenue, as a function of empirical bidder behavior.

Mechanism design has traditionally been largely an analytic process. Assumptions such as full rationality are made about bidders, and the resulting properties of the mechanism are analyzed in this context [76]. Even in large-scale real-world auction settings such as the FCC Spectrum auctions, game theorists have convened prior to the auction to determine the best mechanism to satisfy a set of objectives. Historically, this process has been incremental, requiring several live iterations to iron out wrinkles, and the results have been mixed [30, 109]. An important component of this incremental design process involves reevaluating the assumptions made about bidders in light of auction outcomes. In particular, these assumptions pertain to bidders' intrinsic properties and to the manner by which these properties are manifested in bidding strategies. In general, assumptions are often made about

- Bidders' motivating factors such as valuation distributions and risk aversion;
- Information that is available to the bidders; and
- Bidder rationality.

Even when the assumptions about bidders can be successfully revised based on their past behavior, the process requires human input and is time consuming, undermining the efficiency with which changes can be made to the mechanism.

In the case of the FCC spectrum auctions, months or years elapse between each iteration, leaving time for the experts to reconvene and update the mechanism. But in e-commerce settings in which a large number of auctions for similar goods may be held within a short time frame, such as auctions on eBay or keyword auctions, this inefficiency is a serious drawback. Perhaps the biggest challenge results from the fact that, in practice, bidders are not able to attain full rationality in complex, real-world settings [49]. Rather, they employ heuristic strategies that are in general opaque to the seller, certainly a priori, and often even after the auction.

One method of addressing these challenges that has received recent attention is the use of machine learning algorithms to revise auction parameters in response to observed bidder behavior. For instance, [16] and [17] consider the problem of maximizing seller revenue in a specific type of online auction (similar to a posted price mechanism) through the use of online learning algorithms for combining "expert" advice. Such approaches differ significantly from the traditional approach to mechanism design in that few or no assumptions are made about bidders, and auction mechanisms are evaluated based on worst-case performance.

In this dissertation, I discuss somewhat of an intermediate approach. In this chapter, I present an *adaptive mechanism* that changes in response to observed bidder behavior through the use of a learning algorithm, similar to the methods described in the previous paragraph. In particular, I present an adaptive mechanism in which a continuous parameter (or parameters) defining the mechanism is dynamically adjusted over time, thus enabling a parameter optimization approach. In the next chapter, however, I will assume that reasonable predictions about a *range of possible bidder behaviors* can be made (possibly through previous experience in different markets), and I will show how to choose the learning algorithm in such a way that *expected* performance is optimized with respect to these predictions.

2.2 Adaptive Auction Mechanisms

To motivate the problem under consideration, I begin with an illustrative example that I will refer to throughout the next two chapters. Consider the challenge faced by a seller auctioning off items through an auction service such as eBay. For each auction, the seller must set various parameters defining the particular auction mechanism to be used. (In the case of eBay auctions, these parameters include the start time and duration, start price and reserve price, and possibly a buy-it-now price.) The behavior of bidders depends heavily on the type of item being sold; for instance, buyers of rare coins and buyers of video games are likely distinct populations with very different approaches to bidding. As a result, the mechanism that works best for one type of item may not be appropriate for a different item.

The goal of the seller is to identify the auction parameters that will result in the highest revenue for each item sold. When extensive historical data on past auctions of identical items is available (as is the case with eBay), it may be possible for the seller to estimate the optimal parameters by analyzing this data (e.g., [92]). This approach is not always possible, however. If the seller is introducing a new item to the market, no such data will be available. Alternatively, if there is a sudden change in demand for an item, past data may not accurately reflect the behavior of current bidders. In such cases, the seller must guess which auction parameters will work best. If the seller has multiple identical items to sell, however, then it may be possible for the seller to learn from its own experience about the effectiveness of various parameter values.

This situation illustrates the general problem considered in this chapter. While the effectiveness of an auction mechanism can vary drastically as a function of the behavior of bidders, this behavior is often difficult to predict when choosing the mechanism. In settings in which a large number of similar auctions are held, it may be reasonable to assume that although bidders' behaviors may change, they remain somewhat consistent for a period of time, suggesting the possibility of learning about bidder behavior through experience. For example, the bidders on a particular Google keyword may remain the same for some time, and identical items on eBay will likely attract similar buyers. For such settings, this chapter proposes an *adaptive auction mechanism*, an online empirical process whereby the mechanism adapts to maximize a given objective function based on observed outcomes. Because we allow for situations in which bidder behavior cannot be predicted beforehand, this process must be performed online during interactions with real bidders. (In this chapter and the next, the term "online" refers to the fact that adaptation takes place during the course of actual auctions, and not the fact that auctions take place electronically–although that may also be the case.)

Our view of adaptive mechanism design is illustrated in Figure 2.1. A parameterized mechanism is defined such that the seller can use an *adaptive algorithm* to revise parameters in response to observed results of previous auctions, choosing the most promising parameters to be used in future auctions. Upon execution, the parameterized mechanism clears one or more auctions involving a population of bidders with various, generally unknown, bidding strategies. The results of the auction are then taken as input to the adaptive algorithm as it revises the mechanism parameters in an effort to maximize an objective function such as seller revenue. Any number of continuous or discrete auction parameters may be considered, such as reserve prices, auctioneer fees, minimum bid increments, and whether the close is hard or soft. (For an extensive parameterization of the auction design space, see [112].)

The adaptive algorithm is essentially an online machine learning algorithm aiming to characterize the function from mechanism parameters to expected revenue (or any other observable objective function). Because the seller can select its own training examples (by choosing which set of auction parameters to try next), and because the target output is, in general, continuous, the problem is an active

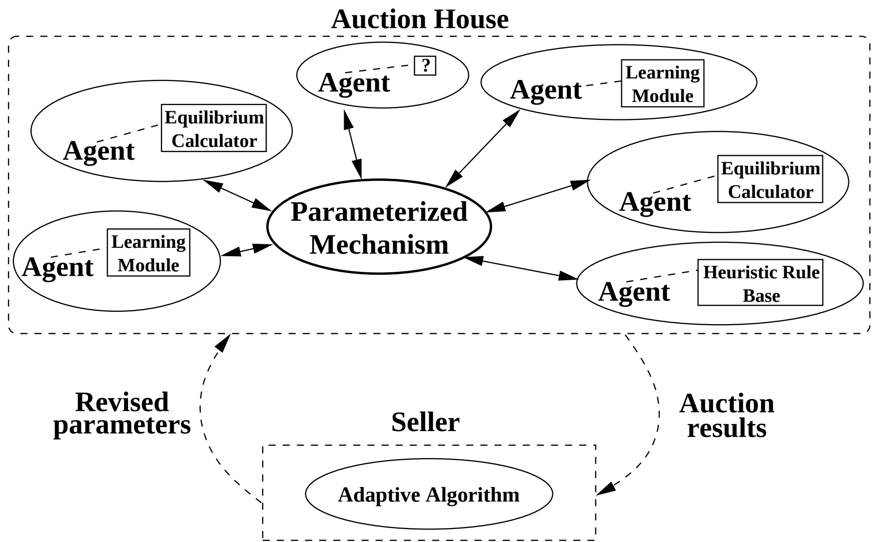

Figure 2.1: A high-level illustration of the concept of adaptive auction mechanisms. From the point of view of the seller, the bidder behaviors are unknown aspects of the environment.

learning [85] regression problem. A key characteristic is that the learning is all done online, so that excessive exploration can be costly.

The bidders in Figure 2.1 may use a variety of different bidding strategies, including heuristic, analytic, and learning-based approaches. For the latter to make sense, the same bidders must interact repeatedly with the mechanism, leading to a potential co-evolutionary scenario in which the bidders and mechanism continue to adapt in response to each other [77]. However, our approach does not depend on repeated interactions with the same bidders. The only required assumption about the bidders is that their behavior is somewhat consistent (e.g., bidders associated with a particular industry tend to bid similarly) for a sufficient period of time to allow for prediction of auction results as a function of the mechanism, at least in expectation.

In the remainder of this section, I formally state the general learning problem the seller faces and introduce an appropriate adaptive algorithm to use in an adaptive auction mechanism.

2.2.1 Learning Problem

Consider a seller who has n identical items to be auctioned off in a series of sequential auctions, which will be referred to as an *episode*. The seller's objective is to maximize its total utility over the course of an episode. The seller's utility may defined in terms of any observable auction outcome of interest, such as the seller's revenue or bidder participation. Henceforth, it is assumed that the seller's objective is to maximize total revenue over the course of the episode, unless otherwise stated. To improve its revenue, the seller may revise the parameters of the auction mechanism. We assume that no prior knowledge is available that would enable complete specification of the bidding strategies individual bidders will exercise. In the absence of an explicit formulation of bidders' behaviors, we are interested in learning a function F that maps auction parameters from a space X to the expected revenue (in $\mathbb{R}$). The seller is able to learn the function over time by experimenting with different parameter values.

For simplicity of exposition, assume for now that that the auction mechanism is determined by a single, discrete parameter with k possible values (i.e., X is a finite set and $|X| = k$). The resulting problem can be mapped to the multi-armed bandit problem, a classic reinforcement learning problem [102]: a gambler has to decide which of k slot machines to play in a sequence of trials so as to maximize the overall reward. Each unique parameter value chosen for an auction during the episode corresponds to a slot machine in the bandit problem. Because the expected revenues from different parameter values (i.e., F) are not known a priori, these revenues can only be learned through experience. Given the seller's objective, consider the following two extreme strategies. The first is a greedy *exploitation* strategy, where for each auction the seller selects the parameter value estimated at that time to be best. If the seller's estimation is accurate, this strategy guarantees maximizing the seller's expected revenue. However, when the seller's estimation is imprecise, this strategy might select suboptimal parameter values. Also, an exploitation strategy will not result in additional information about parameter values other than the one currently believed to be best. The other extreme approach is pure *exploration*: selecting different parameter values throughout the course of the episode so as to improve the seller's estimation of F as much as possible. Such sequential exploration may help to ultimately identify the optimal parameter value, but this strategy can also be costly to the seller because many suboptimal parameter values may be tried repeatedly, resulting in low total revenue for the episode. Thus, when selecting the parameter value for each auction, the auctioneer faces a tradeoff between exploring to improve its estimation and and exploiting the parameter value currently estimated to be best.

2.2.2 Bandit Adaptive Algorithm

To manage this tradeoff, the seller must choose an adaptive algorithm to determine how the parameter value should be selected in response to prior auction outcomes. One algorithm commonly used for bandit problems is *softmax action selection* [102] using the Boltzmann distribution, which combines the exploitation and exploration strategies discussed above. In this approach, the average result for each choice of parameter value x, avg_x, is recorded, and at each step the probability of choosing x at time step t is given by $(e^{avg_x/\tau_t})/(\sum_{y=1}^{k} e^{avg_y/\tau_t})$, where τ represents a *temperature* determining the extent to which exploitation trumps exploration. Note that the larger the temperature, the more the resulting distribution will resemble a uniform distribution, encouraging exploration. As a result, the temperature is often lowered over time to favor increasing exploitation due to the fact that estimates of the revenue from each choice improve in accuracy with experience. To maintain a record of the average revenue for each choice, for each choice we must track both the average revenue so far, avg_x, and the number of times that choice has been tried, $count_x$.

In order to implement softmax action selection, we must choose values for each τ parameter and initial values for each *avg* and *count*. We vary the temperature throughout an episode by choosing starting and ending temperatures, τ_{start} and τ_{end}, and interpolating linearly. Although the straightforward approach to initializing averages and counts would be to set them all to zero, one common technique, known as *optimistic initialization* [102] is to set all counts to a small value and all initial averages to a value higher than the predicted highest possible result. Each choice is therefore likely to be explored at least once near the beginning of the episode. I will refer to these initial values as *prior experience*, as they have the same effect as beginning with a certain set of experience. (They also serve a role somewhat similar to a Bayesian prior.) For the experiments of this chapter, all of these values will be chosen through limited experimentation to give reasonable performance. In the next chapter, I will show how these values can be tuned more effectively when additional information about bidders is available. Pseudocode for this adaptive algorithm, which will be referred to as the *bandit adaptive algorithm*, is shown as Algorithm 1.

2.2.3 Regression Adaptive Algorithm

The approach just described is limited by the fact that it can only handle a discrete set of auction parameters. In addition, it assumes that the expected revenue from each choice is independent, meaning that information obtained about one parameter value tells us nothing about other parameter values. In reality, many of the auction parameters we might wish to tune are continuous. As a result, it is likely that the expected revenues of nearby choices will be similar, and thus experience could be profitably shared between choices. To address this issue, I now

Algorithm 1 `Bandit Adaptive Algorithm`

Input a finite set X of size k of possible auction parameter values, the number of auctions to run n, τ_{start} and τ_{end}, $\forall x \in X : count_x$ and avg_x

For $0 \leq t < n$:

1. $\tau_t = \tau_{start} + (\tau_{end} - \tau_{start}) \cdot \frac{t}{n}$
2. Draw a parameter value from the distribution P(x) = $(e^{avg_x/\tau_t})/(\sum_{y=1}^{k} e^{avg_y/\tau_t})$.
3. Run an auction and obtain outcome o.
4. $avg_x \leftarrow (avg_x \cdot count_x + o)/(count_x + 1)$
5. $count_x \leftarrow count_x + 1$

introduce an enhanced approach that I will call the *regression adaptive algorithm*. As the name suggests, we perform regression over past auction results to derive a function mapping the parameter value to the expected revenue. In particular, we perform locally weighted quadratic regression (LWQR) [3], a form of instance-based regression. This approach combines the simplicity of classical least-square regression with the flexibility of non-parametric modeling. To predict the expected revenue for a given choice of parameter value x, the weight of each existing data point (i.e., previous auction outcome) in a data set D is determined by taking its distance d from x and applying a Gaussian kernel of width w: e^{d/w^2}. Parameters are then found specifying the quadratic that minimizes the weighted sum of squared errors, and the quadratic is then evaluated at x. This process is repeated for each choice for which we want an estimate of expected revenue.

Because we can now predict the expected revenue of any choice of parameter value, even if we have no experience at that choice, we are no longer restricted to considering a finite number of choices as in the bandit approach. We continue to discretize the range of choices for computational reasons–doing so allows us to implement an incremental version of LWQR and also to use softmax action selection without modification. However, we are able to effectively use much finer discretizations than before. In fact, in the experiments below we observed no benefit from increasing beyond 100 choices, so we treat the degree of discretization as a fixed parameter for the regression approach, and reinterpret k as described below.

The parameters for the regression adaptive algorithm are almost the same as those of the bandit adaptive algorithm. We allow the temperature to vary as before. The concept of optimistic initialization remains similar. As we are now dealing with a continuous parameter space, we may choose to begin with prior experience at any auction parameter values we wish. We choose parameter values p_i for $0 \leq i < k$ that are uniformly spaced values over the range of the parameter. We then populate the initial data set D with k tuples of the form $(p_i, avg_i, count_i)$, where avg_i refers an outcome which has been observed for parameter value p_i, and $count_i$ refers to the number of times that outcome has been observed. When LWQR is performed, the weight of each data point is multiplied by $count$. As we observe real auction outcomes, we add additional tuples to D. It should be noted that for the regression adaptive algorithm, k is used only to specify the number of points used as prior experience, and is independent of whatever degree of discretization is used for selection of auction parameter values. The only additional learning parameter is the kernel width w used in the weighting function. Observe that this approach generalizes easily to settings with multiple continuous auction parameters, as LWQR can easily be extended to multiple regression. Pseudocode for the regression adaptive algorithm is shown as Algorithm 2.

2.3 Auction Settings

In this section I introduce two distinct auction settings that will be used as experimental domains. These settings are distinguished by both the auction parameter that is to be tuned and the behavior of bidders. Both settings involve

Algorithm 2 `Regression Adaptive Algorithm`

Input a continuous parameter space X and a discretization of this space X', the number of auctions to run n, τ_{start} and τ_{end}, kernel width w, and an initial data set D

For $0 \leq t < n$:
1. $\tau_t = \tau_{start} + (\tau_{end} - \tau_{start}) \cdot \frac{t}{n}$
2. Apply LWQR to data set D with kernel width w to induce the mapping $F : X \rightarrow \mathbb{R}$
3. Draw a parameter value x from the distribution P(x) $= (e^{F(x)/\tau_t})/(\sum_{y \in X'} e^{F(y)/\tau_t})$.
4. Run an auction and obtain outcome o.
5. Add the new experience to the data set: $D \leftarrow D \cup (x, o, 1)$

an English (ascending, open-cry) auction in which the bidders have independent, private (i.e., unknown to other bidders) values for the goods being sold. Bidders submit ascending bids until no incremental bids are made above the winning bid. Again, the seller auctions the n identical items one at a time to bidders from a given population through a series of auctions.

2.3.1 Reserve Price Setting

The first setting involves setting a reserve price for each auction indicating the minimum acceptable bid. In the absence of any bid higher than the reserve price, no transaction occurs. It has been shown that when bidders are rational, the optimal reserve price should be higher than the seller's valuation of the item [73]; however, a reserve price of 0 is often seen in practice.

Dodonova and Khoroshilov explain this phenomenon by bidders' *loss aversion* [36]. Loss aversion violates the rationality assumption because the utility from a gain is lower than the disutility from a loss of the same magnitude. Specifically, if the marginal utility from winning an auction is x, then the marginal disutility from losing the same object is αx, where $\alpha > 1$. A bidder considers that it is "losing" an item if it was the high bidder at some point in the auction, but then does not win the item. Thus, in practice, a loss-averse bidder bids more aggressively after having had the highest bid at any point during the auction.

We assume that the bidders are, to varying degrees, loss averse. Note that if $\alpha = 1$ we arrive at the traditional loss neutral bidders as a degenerate case. Under these assumptions and model setup, Dodonova and Khoroshilov derive the equilibrium as follows. Assuming two loss averse bidders, a first mover submits a bid in the beginning of the auction if its valuation is higher than the reserve price. The second bidder responds by submitting an increment above the current winning bid only if by doing so the bidder can guarantee a positive expected utility. In particular, this will be the case only if

$$\int_{r}^{v_2} (v_2 - \alpha v_1) f(v_1) dv_1 > 0$$

where r is the reserve price, v_2 is the second bidder's valuation, and f is the (known) probability distribution function over valuations. Intuitively, the second bidder recognizes that loss aversion may lead it to pay more than his valuation if the first bidder's valuation is sufficiently high, making it more reluctant to enter the auction. With only one active bidder, the auctioneer's revenue is decreased.

If all bidders participate, the auction continues as a standard ascending price English auction until a bidder's marginal utility from losing the object is less than the (potential) winning bid, i.e., the losing bidder will bid up to α times its valuation and then drop out. This equilibrium can cause the seller's optimal reserve price to be 0 under certain conditions. For instance, if f is a uniform distribution, a reserve of 0 will maximize the seller's revenue for values of α above 1.3. The equilibrium can also result in a non-convex revenue as a function of reserve price, with one maximum close to zero and another at a much higher reserve price, as shown in Figure 2.2. Thus the auctioneer has potential incentives to set both a low reserve price and a high reserve price, a conflict that must be taken into account when choosing a method of searching for the optimal reserve price.

In this setting, the seller interacts repeatedly with bidders drawn from a fixed population (characterized by distributions over valuations and α). For the sake of simplicity and to allow the use of the equilibrium strategy as presented in the previous section, we assume that exactly two bidders participate in each auction, and that these bidders participate in no other auctions (The next auction setting described will relax these assumptions.) The seller sets a reserve price for each auction, thus restricting the possible bids available to the bidders and indirectly affecting the auction's outcome. The seller's goal is to set the reserve price for each auction so that the total revenue obtained from all the auctions is maximized. If a complete model of the behavior of the population of bidders were available, the seller could determine the optimal reserve price analytically by solving for the reserve price maximizing expected revenue under this model. However, we assume that the seller does not have such a complete model, perhaps because it is a new item. Thus, the seller must identify the optimal reserve price through online experimentation guided by an adaptive mechanism.

A bidder is characterized by i) an independent, private value v for the sold item, and ii) a degree of loss-aversion α. A given bidder assigns the same value to any one of the items sold, and the population of bidders does not change over time. Thus, the behavior exhibited by bidders is the same for each auction *in expectation*, allowing the seller to draw inferences from past auction results.

In our experiments, we are interested in evaluating our adaptive auction mechanism over a wide range of different bidder populations. We therefore follow a method of generating bidder populations that amounts to specifying a probability distribution over populations and drawing a population from this distribution for each episode to be simulated. We generate an individual bidder population in our experiments as follows. For each episode, we randomly generate a distribution for valuations by taking a Gaussian with a mean chosen uniformly from $[v_{min}, v_{max}]$ and a variance of 10^x with x chosen uniformly from [-2, 1], and then normalize the distribution so that the portion over the range $[v_{min}, v_{max}]$ represents a PDF. We then generate a distribution for α in the same way, choosing variance as before and using a range of $[\alpha_{min}, \alpha_{max}]$ for both the mean and the entire distribution. For the experiments of this chapter, we use the values $n = 1000$ auctions, $v_{min} = 0$, $v_{max} = 1$, $\alpha_{min} = 1$, and $\alpha_{max} = 2.5$.

We simulate bidder behavior by having bidders follow the equilibrium strategy described above under the assumption that the other bidder has the same α (because this is the situation to which the equilibrium solution applies). Thus for each auction in an episode, we draw two values from the valuation distribution, draw a single α from the α distribution, randomly assign one bidder to be the initial bidder, and then have both bidders bid as specified in the equilibrium strategy.

Figure 2.2 helps illustrate the task faced by the seller. The left side shows the average revenue as a function of reserve price for three different populations. (The mean of v, variance of v, mean of α, and variance of α are 0.8, 0.01, 1, and 0.1 respectively for population A; 0.3, 1, 1.75, and 1 for population B; and 0.4, 0.1, 1.5, and 0.01 for population C.) In each case, the function has a different shape and a different maximum. On the right side of Figure 2.2, average results are shown for 10,000 bidder populations, generated randomly as described above. The solid line represents the average revenue for each choice of reserve. A reserve price of 0.54 yields the highest average revenue, 0.367. If we were required to select a single reserve price for the seller to use, we would choose this price. However, we know that for each individual bidder population there is a distinct choice of reserve that yields the highest average revenue. In particular, the dotted line shows the number of times that each reserve (tested at intervals of 0.01 between 0 and 1) was optimal. Two important observations can be made: i) despite the variety in bidder

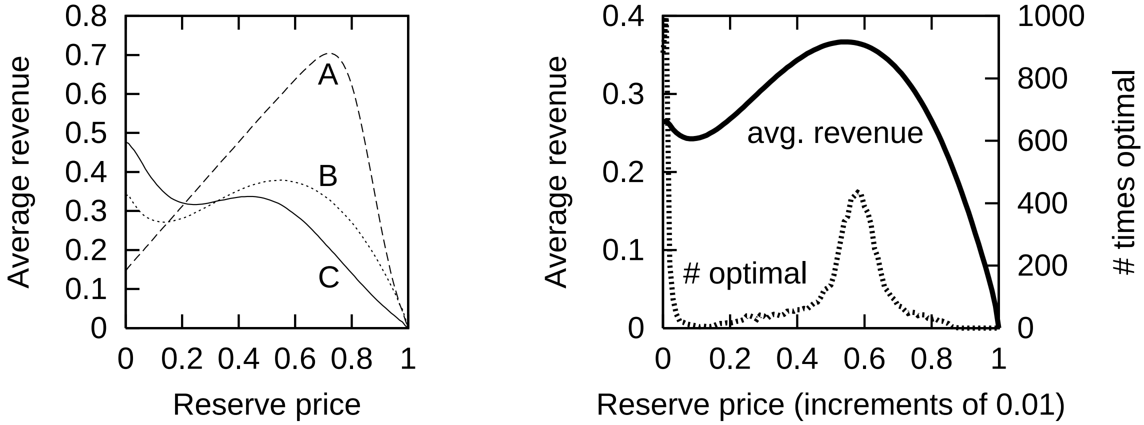

Figure 2.2: The graph on the left shows the average revenue for each reserve price for three different populations. The graph on the right shows results for 10,000 populations. The solid line represents the average revenue for each reserve price, while the dotted line represents the number of times each price was optimal.

populations, the optimal reserve price is frequently in one of two small regions (including near zero, as is expected with loss averse bidders); ii) nevertheless, most choices of reserve are optimal for some population. It is the second observation that motivates our use of an adaptive mechanism.

2.3.2 BIN Price Setting

The previous auction setting involved the simplifying assumptions that exactly two bidders participate in each auction, that these bidders participate in no other auctions, and that they bid according to a known equilibrium. In the second auction setting, these assumptions are relaxed. As before, we consider English auctions in which bidders have independent private values. However, we now allow bidders to arrive at any time during an auction and to keep bidding in auctions until they either win or choose to drop out. In addition, both bidders and the seller are now time-sensitive (but not loss averse). Instead of choosing a reserve price, the seller now chooses a buy-it-now (BIN) price at which a bidder may claim the item at any time, ending the auction. (The BIN price remains valid throughout the auction, in contrast to some implementations of BIN auctions.) Setting the BIN price too low can reduce revenue from the auction, while setting it correctly may actually induce impatient bidders to accept a BIN price higher than the price they might pay if the auction ran to completion. Finally, we assume the seller has only 100 items to sell ($n = 100$), and that a new auction begins as soon as the previous one has completed.

To simulate auctions in continuous time, we divide each auction into 10 periods and allow bidders to submit or revise bids once per period, with all bids submitted simultaneously and processed in random order. In each period, the number of new bidders arriving is drawn from a Poisson distribution with parameter λ representing the expected number. Each bidder's type consists of a valuation v in [0,1] and a cost per period c in [0.0001, 0.001]. This cost represents a participation cost (actual or perceived) paid by the bidder for each period it remains active. The seller also has a cost per period of 0.02, giving it an incentive to set the BIN price low enough to shorten auctions. The final element needed to run our auction simulations, the strategy used by bidders, is described below.

To generate a bidder population, we first choose λ uniformly from [0.5, 2]. We then choose a distribution over valuations by taking a Gaussian with a mean chosen uniformly from [0.2, 0.8] and a standard deviation chosen uniformly from [0.05, 0.3], and normalizing the distribution so that the portion restricted to the range [0, 1] represents a PDF. Finally, we choose a distribution over costs per period by taking a Gaussian with a mean chosen uniformly from [0.0002, 0.0009] and a standard deviation chosen uniformly from [0.0001, 0.0005], and normalizing the distribution so that the portion restricted to the range [0.0001, 0.001] represents a PDF. As before, this method of generating bidder populations can be seen as implicitly defining a distribution over populations.

In the previous auction scenario, we assumed that bidders behaved in accordance with a known equilibrium. We now demonstrate that in the absence of theoretical or empirical knowledge about bidder behavior, it is possible to use learning to generate the strategy to be used by bidders in simulation. In particular, we use a process of competitive coevolution to evolve neural networks that take the information available to a bidder as input and output the actions of the bidder. The eleven inputs to the network represent the following information: the current winning price, whether the bidder is currently winning, the number of periods left, whether this is the last period, the total number of periods in which the bidder has participated, the BIN price, the winning price of the previous auction, the average and standard deviation of the last five winning prices, and the bidder's valuation and cost per period. The two outputs are the bid to place (accepted only if it is higher than the previous bid) and whether the bidder wishes to stop participating in auctions (the current winner cannot stop participating, and the final winner of each auction automatically stops). The networks are evolved using NeuroEvolution of Augmenting Topologies (NEAT) [97], a genetic algorithm for evolving both the topologies and weights of neural networks. We use the ANJI 2.0 implementation of NEAT with default parameters. The process is initialized with 500 random neural networks representing the first generation. In each generation, a fitness score for each network is determined by having the network participate in a number of auction episodes, as described below. Then a new generation of 500 networks is created by combining pairs of networks from the current generation (known as crossover) and then slightly modifying the results (mutation). A few of the top networks are copied unchanged into the new generation (elitism). The probability of a network being chosen to "reproduce" during this step is determined primarily by its fitness score. This process continues for 500 generations, although the behavior of the top networks tends to stabilize in around half of this time. At the end, we chose one network to represent the bidding strategy that will be used in our experiments: one that was successful over the last several generations, rather than simply the best performer in the final generation.

Within each generation, we determine the fitness of each network as follows. We begin by simulating 500 episodes, drawing a different bidder population in each episode. We simulate auctions using a non-adaptive seller that simply chooses auction parameters (the BIN price) randomly from a reasonable range, ensuring that networks are tested under a variety of situations. (The next chapter will explore the effects of using an adaptive seller during the evolutionary process.) Each time a new bidder arrives, it is randomly assigned a network as its bidding strategy. At the end of the 500 episodes, for each network we determine the average utility of all bidders to which it was assigned. While we could stop at this point, instead we remove those networks that had an average utility below the average of all networks, and with the remaining networks we perform another round of 500 episodes. We continue this process of elimination until fewer than 5 networks remain. Each network is then assigned a fitness as follows: if the network reached the nth round of this process, and had an average utility u in that round, its fitness is given by $10^{(n-1)} + 10^n u$. The reason for performing multiple rounds is that many of the networks in each generation can have very poor performance. The first few rounds may therefore represent situations in which the behavior of bidders is unrealistic, and so we should not place much emphasis on the performance of a network in these rounds. Instead, we favor networks that can survive the early rounds and then perform well in the final round, against other strong networks. In fact, by the final round, it is possible that several simultaneously active bidders may be using the same network, and it is important to evaluate networks in this situation given that we use a single network in our experiments. Finally, note that simply evaluating each network against itself is not a valid alternative, as collusion becomes possible - a network might always bid zero and obtain a high average utility. Our use of competitive coevolution prevents such a network from

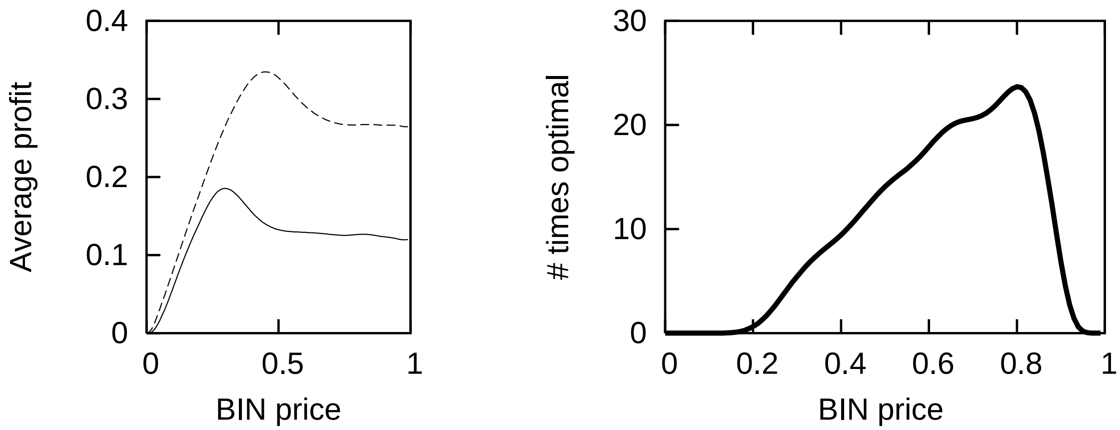

Figure 2.3: The graph on the left shows the average profit for each BIN price for two different populations. The graph on the right shows the number of times each price was optimal over 1000 populations.

performing well.

The network chosen to represent bidder strategies appears to result in reasonable behavior–bidders with low valuations drop out early, highly time sensitive bidders often accept the BIN option, etc.

Figure 2.3 shows how the BIN price can affect profits for different bidder populations. The graph on the left shows the average profit (the seller's revenue minus its time cost) when each price is used in all 100 auctions for two different populations. The graph on the right shows how often each BIN price was optimal out of 1000 randomly generated populations. Most prices were optimal for at least one population, illustrating the need for learning.

2.4 Experiments

This section presents the results of experiments in the two auction settings described above. For both settings, the results of adaptive auction mechanisms are compared against using the best fixed auction parameter for the distribution over bidder populations used in the simulations (which would not actually be known to the seller).

2.4.1 Reserve Price for Loss Averse Bidders

For the setting in which the seller must set a reserve price and interacts with loss averse bidders, we implemented both the bandit and the regression adaptive algorithms. For both algorithms, we set $k = 13$. In the case of the bandit approach, this meant that 13 discrete auction parameter values could be used by the seller, and so we chose 13 uniformly spaced values over the range of the reserve price, i.e., $[0, 1]$. We used these same values for the reserve prices of the prior experience (initial data points) for the regression approach. For averages and counts, we used a somewhat optimistic value of 0.6 for each avg_i and a value of 1 for each $count_i$. τ_{start} and τ_{end} were set to 0.1 and 0.01, respectively, and for the regression approach a kernel width of 0.1 was used. With the exception of k, all of these learning parameters were chosen through limited experimentation to give reasonable performance–these represent parameters that a knowledgeable practitioner with little prior information about the setting might choose. As the value of k is especially important to the performance of the bandit approach (because it restricts the auction parameter values available to the seller), we experimented with many values of k and found 13 to be optimal for the bandit approach given the other learning parameters chosen. The regression approach was much less sensitive to the choice of k, as would be expected. As a result, these experimental conditions are relatively favorable for the bandit approach.

We measured the average revenue per episode for both adaptive algorithms over 10,000 episodes. A different bidder population was drawn for each episode. For each approach tested, the random number generator used in generating all bidder parameters received the same seed, ensuring identical bidder behavior. The average revenues per auction are shown in Table 2.1, while a plot of the average

Adaptive algorithm	Avg. revenue
best fixed reserve price (0.54)	0.367
bandit adaptive algorithm	0.374
regression adaptive algorithm	0.385

Table 2.1: Average revenue per auction for each adaptive algorithm over 10,000 populations. Differences are statistically significant at the 99% confidence level according to paired t-tests.

revenue for each auction over an entire episode is shown in Figure 2.4. The average total revenue in each case is higher than the revenue resulting from using the best fixed reserve price, 0.54, indicating that the use of an adaptive mechanism is indeed worthwhile in this scenario. The difference observed between each pair of approaches is statistically significant at the 99% confidence level according to paired t-tests comparing results for the same bidder population. From Figure 2.4 we can see that both adaptive algorithms approach the same revenue by the last auction in an episode, suggesting that being limited to a discrete set of auction parameter values is not a serious disadvantage to the bandit approach. However, the regression approach performs much better early on, suggesting that the main drawback to the bandit approach is its implicit assumption about the independence of different parameter values. In fact, the regression adaptive algorithm requires far fewer than 1000 auctions to be beneficial (i.e., obtain a higher revenue sum than the fixed reserve).

2.4.2 BIN Price for Time Sensitive Bidders

For the setting in which the seller must set a BIN price and interacts with time sensitive bidders, we implemented the regression adaptive algorithm using optimistic initial parameters ($k = 10$, $\tau_{start} = 0.01$, $\tau_{end} = 0.001$, each $count_i = 1$, each $avg_i = 1$, and kernel width 0.1). Again, these learning parameters were chosen through limited experimentation to give reasonable performance. We generated

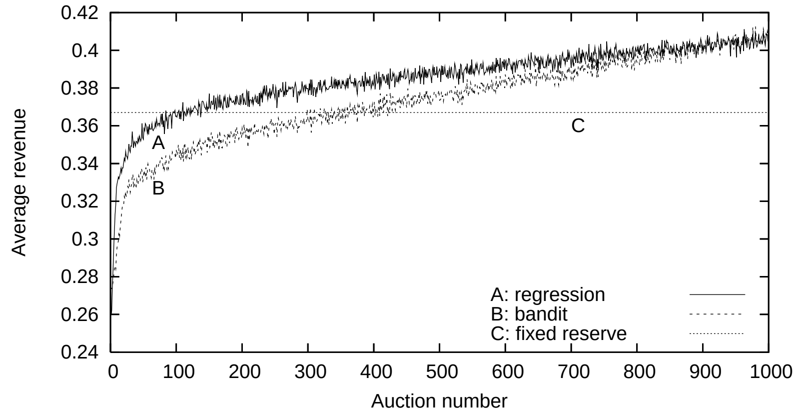

Figure 2.4: Average revenue per auction over the course of an episode for each adaptive algorithm when setting reserve prices.

10,000 bidder populations and evaluated both the regression adaptive algorithm and the fixed BIN price with the highest expected profit (0.73). Figure 2.5 shows the average profit for each auction for each approach. Overall, the average profit per auction was 0.4884 for the fixed BIN price and 0.4711 for the regression adaptive algorithm, and these differences are statistically significant with 99% confidence according to paired t-tests. The regression adaptive algorithm was eventually able to identify a good BIN price, but its early exploration was so costly that it underperformed the fixed price overall. However, if the number of items to auction off were higher, as in the previous set of experiments for the reserve price setting, then using an adaptive auction mechanism would be worthwhile in this setting as well. It should also be noted that the best fixed BIN price is not in fact known to the seller.

2.5 Summary

This chapter introduced the concept of adaptive auction mechanisms for sellers who wish to auction off many identical items sequentially. An adaptive auction mechanism makes use of an adaptive algorithm to revise auction parameters

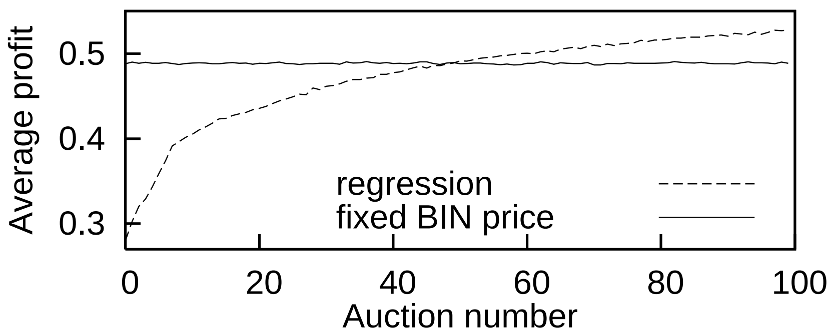

Figure 2.5: Average revenue per auction over the course of an episode when setting BIN prices.

in response to observed bidder behavior in order to maximize an objective such as total revenue. The main contribution of this chapter is the regression adaptive algorithm, which can learn continuous auction parameters online using a combination of the softmax action selection algorithm from the reinforcement learning literature and Locally Weighted Quadratic Regression. In two distinct auction settings characterized by different auction parameters and bidder behaviors, this approach was able to learn auction parameters that were better than the best fixed parameter value. However, it often took many auctions to reach that point, and a number of learning parameters need to be set to ensure reasonable performance. In the next chapter, I show how both of these issues can be addressed if the seller has some amount of prior knowledge about bidder behavior.

Chapter 3

Adaptive Auction Mechanisms with Prior Knowledge

Chapter 2 introduced the idea of adaptive auction mechanisms. In this chapter, I show how prior knowledge of bidder behavior can be used to improve the performance of an adaptive auction mechanism. After the nature of this prior knowledge is specified in Section 3.1, Section 3.2 presents two improvements to the regression adaptive algorithm of Chapter 2 that can make use of this knowledge, fulfilling Contribution 3 for the sequential auction scenario as outlined in Section 1.2. Section 3.3 contains experiments testing these algorithms under various conditions, fulfilling Contribution 4.

3.1 Prior Knowledge of Bidder Behavior

There are a number of forms that prior knowledge of bidder behavior could take. As one illustrative example, suppose that our seller has a large number of copies of a newly published book to be sold through a series of auctions. As the books are new, there are no previous auction results available to guide the choice of auction parameters. The seller is not completely in the dark, however, because auction results for books by similar authors or on similar topics are available, and these books likely attract bidder populations with characteristics similar to those of potential bidders on the new book. If the seller wanted to choose a single set of auction parameters, a reasonable choice might be the parameters that were best for the most similar book, or an average of the best parameters for several similar books. If instead the seller wishes to use an adaptive mechanism and needs to choose the adaptive algorithm for it, it makes similar sense for the seller to choose an adaptive algorithm that *would have worked well* for these similar books. To determine how well an adaptive algorithm would have worked for a particular book, the seller could attempt to simulate the behavior of the population of bidders for that book using the past auction data, and apply the adaptive algorithm to these simulated bidders.

Alternatively, suppose that the seller receives marketing information suggesting valuations that buyers might have for the book, along with information suggesting bidding strategies that bidders tend to use in such auctions. Again, if the seller wishes to use an adaptive auction mechanism, it makes sense to choose an adaptive algorithm that would work well if this information is correct. The seller could again evaluate an adaptive algorithm by simulating a number of bidder populations that are plausible given the information available and applying the algorithm to each population.

In both cases, we have a seller that wishes to choose an adaptive algorithm in order to implement an adaptive auction mechanism, and the seller is able to make predictions concerning possible bidder behavior that allow it to simulate a number of possible bidder populations. The goal of the seller is to choose an adaptive algorithm that performs well with the actual bidder population it encounters, but as this population is unknown in advance, the seller attempts to identify an adaptive algorithm that performs well under simulation. Therefore, in this chapter, prior knowledge will be represented as the ability to simulate some plausible range of bidder behavior. In fact, this ability amounts to specifying a probability distribution over bidder populations, as was done in the experiments of Chapter 2. One goal of the experiments in this chapter will be to see how much the performance of the proposed adaptive algorithms depends on the similarity between the distribution over *simulated populations* and the distribution over actual *encountered* *populations*. (The terms "simulated" and "encountered" will be used throughout this chapter to distinguish the two types of populations.)

3.2 Algorithms

In Section 1.1 of the introduction, I outlined four general approaches to adaptation that could be taken when previous market experience–in this scenario, the prior knowledge of bidder behavior–is available. The first approach, learning from only the new market only, was covered in Chapter 2. Below, I describe two algorithms representing implementations (and in one case, a combination) of the remaining three approaches. These algorithms represent modifications of the regression adaptive approach described in Chapter 2.

3.2.1 Metalearning

One of the general approaches to using prior knowledge is to learn from both the simulated and encountered markets. In Section 1.1, this approach was subdivided into two possibilities: combining data, or combining models. For the regression adaptive approach, it turns out that no distinction to the two needs to be made, because we are using an instance-based learner–LWQR–to perform the regression, and in essence, the data *is* the model. The remaining question thus becomes *how* to combine data generated during simulation with data generated from the encountered population. Simply generating a large amount of prior experience (auction parameter values and the resulting outcomes) in simulation and using this with the regression adaptive algorithm is not likely to work well, because this prior experience will dwarf actual experience. As a result, when the regression adaptive algorithm favors exploitation of the parameter value believed to be best, that value will tend to be the value that was best for the distribution of simulated populations, not the encountered population.

An alternative approach is to learn from only the encountered population, but to use prior knowledge to tune the learning parameters. For example, the regression adaptive algorithm requires a kernel width to be set. A reasonable value would be one that worked well when applied in simulation. In fact, this general approach makes sense for setting all learning parameters, and so this approach amounts to finding the set of learning parameters that would have performed best for the distribution over simulated bidder populations. This approach could be viewed as an instance of *metalearning* [107]. In metalearning, the goal is to improve the performance of a learning system for a particular task through experience with a family of related tasks. In our case, the learning system is the regression adaptive algorithm, and the family of related tasks is the set of different bidder populations generated during simulation.

The metalearning approach also resolves the problem of how to handle prior experience, because prior experience can itself be viewed as a parameter to the regression adaptive algorithm. Therefore, metalearning can be used to find the prior experience that results in the best performance in simulation. To illustrate how effective prior experience might be chosen, consider Figure 2.2 from Chapter 2 (shown again here as Figure 3.1). Again, the dotted line shows the number of times that each reserve (tested at intervals of 0.01 between 0 and 1) was optimal over 10,000 simulated populations. Despite the variety in bidder populations, the optimal reserve price is frequently in one of two small regions (including near zero, as is expected with loss averse bidders). A proper choice of prior experience (data points representing either good outcomes in those regions or bad outcomes outside of it) could encourage the regression adaptive algorithm to focus early exploration on those regions.

Determining the set of learning parameters that work best for the distribution over simulated bidder populations can be viewed as an optimization problem.

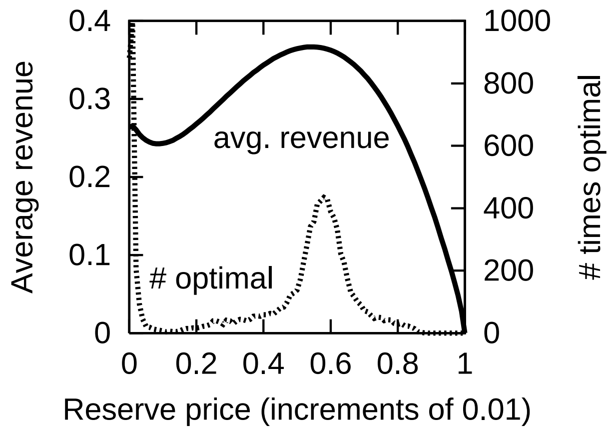

Figure 3.1: Results for different reserve prices for 10,000 populations. The solid line represents the average revenue for each reserve price, while the dotted line represents the number of times each price was optimal.

The objective function to be maximized is the expected revenue (or any other auction outcome) during an episode for that distribution. For any given set of learning parameters, we can obtain an estimate of the expected revenue from an episode by sampling a population of bidders and running an episode using those parameters. This estimate will be highly noisy, due to the large number of random factors involved in the process, and so we are faced with a stochastic optimization task.

To solve this task, we use Simultaneous Perturbation Stochastic Approximation (SPSA) [95], a method of stochastic optimization based on gradient approximation. At each step, two estimates of the expected episode revenue are taken for slight perturbations of the current learning parameters (the same bidder population is used for each estimate), a gradient approximation is found, and the parameters are updated in the direction of the gradient. For reference, all learning parameters for the regression adaptive algorithm are described in Table 3.1. Ideally, the parameter k (representing the number of auction parameter values for which we have previous experience) would be part of the search process as well, but as our search method requires the dimensionality of the search space to be fixed (i.e., the number

Parameter	Description
avg_i, $0 \leq i < k$	average initial revenue at auction parameter value i
$count_i$, $0 \leq i < k$	weight of prior experience at auction parameter value i
τ_{start}	initial temperature
τ_{end}	final temperature
w	kernel width for regression

Table 3.1: Parameters of the regression adaptive algorithm found through metalearning.

of learning parameters must be determined in advance), in each experiment below we have chosen what appears to be the best value of k after running searches for several values.

It should be noted that although this process of searching for the optimal parameters can be time consuming (in our experiments, a number of hours were required for the search to converge), the process takes place in offline simulation before the actual auctions begin. When the regression adaptive algorithm is applied during actual auctions with the encountered population using the resulting learning parameters, each choice of a new auction parameter value takes only a small fraction of a second.

3.2.2 Bayesian Approach

A different general approach to using prior knowledge that was outlined in Section 1.1 is attempting to identify the market from a space of possible markets. In the sequential auction scenario with prior knowledge considered here, that approach amounts to matching the encountered bidder population with one of the bidder populations from simulation. In Chapter 2, bidder populations were simulated by specifying a bidding strategy to be used by all bidders and then drawing specific parameters specifying the type of each individual bidder from some population-specific distribution. Under this setup, identifying a bidder population amounts to determining this population-specific distribution. For example, for the reserve price setting with loss averse bidders described in Section 2.3.1, the behavior of a bidder population is specified completely by four parameters: the mean and variance of the valuations, and the mean and variance of α values. If these four parameters could be determined, it would be possible to directly determine the optimal reserve price for any bidder population encountered.

As identifying the exact population parameters from a small number of auctions is likely not possible, we implement the following approach: using Bayesian inference to maintain a joint probability distribution over these parameters. This joint distribution represents our current beliefs about those parameters as a result of observed auction outcomes. Specifically, we maintain a joint distribution $P(b)$ over possible bidder populations (for convenience of both notation and implementation we assume this is a discrete set). Initially, we set $P(b)$ to our prior, i.e., the distribution over simulated populations. Each observed auction outcome is used to update this probability as follows: $P(b) \leftarrow P(b) \cdot P(o|x, b)/Z$, where $P(o|x, b)$ is the probability that an auction outcome o is observed given auction parameter value x and population b, and where Z is a normalizing constant, to create a distribution in accordance with Bayes' rule.

Bayesian inference allows us to estimate the revenue for any auction parameter value x: $\sum_{b \in B} P(b)O(x, b)$, where $O(x, b)$ is the expected revenue for population b. However, Bayesian inference does not tell us what parameter value to try next in our sequential auction scenario. A number of algorithms exist for action selection in Bayesian settings, but to allow for the most direct possible comparison between Bayesian inference and the adaptive algorithms described previously, we apply softmax action selection using the estimated revenues. Thus, the Bayesian approach is essentially the regression adaptive algorithm with the output of LWQR replaced by our new revenue estimate. The pseudocode for the Bayesian approach is presented in Algorithm 1.

Algorithm 1 `Bayesian Approach`

Input a continuous parameter space X and a discretization of this space X', the number of auctions to run n, τ_{start} and τ_{end}, kernel width w, a discrete set B of possible bidder populations, and a prior distribution $P(b)$ over these populations

For $0 \leq t < n$:
1. $\tau_t = \tau_{start} + (\tau_{end} - \tau_{start}) \cdot \frac{t}{n}$
2. $\forall x \in X'$ compute expected outcome $E(x) = \sum_{b \in B} P(b)O(x,b)$
3. Draw a parameter value x from the distribution Q(x) = $(e^{E(x)/\tau_t})/(\sum_{y \in X'} e^{E(y)/\tau_t})$.
4. Run an auction and obtain outcome o.
5. $\forall b \in B, P(b) \leftarrow P(b) \cdot P(o|x,b)/Z$ for normalizing constant Z

To this point, we have assumed that we have access to the functions $O(x,b)$ and $P(o|x,b)$. However, our bidder model is only a generative one–that is, we have a full model of all relevant factors affecting bidder behavior and can thus sample bidding outcomes, but we cannot analytically express the relationship between the population parameters and the quantities of interest, i.e., outcomes and their probabilities. Additionally, the distributions over bidder populations that we will simulate are continuous, not discrete. To address these issues, we precompute approximations to $O(x,b)$ and $P(o|x,b)$ by discretizing the range of possible population parameters, auction parameters, and outcomes, and using simulation to fill in a table of values. As this process is very computationally intensive, we experiment with the Bayesian approach in only the reserve price setting, and not the BIN price setting.

For the reserve price setting, we use 10 uniformly spaced values for each population parameter (for a total of $10,000$ possible populations), along with 30 uniformly spaced values for both reserve prices and outcomes. For each choice of population and reserve price, the probability that the resulting revenue is closest to each outcome value is estimated through simulation of auctions. Increasing the granularity of discretization beyond this level does not appear to significantly improve the results of applying these approximate probabilities. We represent our beliefs about the bidder population as a probability distribution over all $(10,000)$ possible populations, initialized uniformly in keeping with the method of generating a population described in Section 2.3.1. After each auction, the nearest of the 30 discrete reserve and outcome values are identified, and these are the values used to update the probability distribution.

3.3 Experiments

This section presents the results of experiments using the approaches discussed: the best fixed auction parameter, the regression adaptive algorithm with optimistic parameters (which will be referred to as *optimistic regression*), the regression adaptive algorithm with metalearned parameters (which will be referred to as *metalearned regression*), and the Bayesian approach. These approaches are evaluated under a variety of different circumstances, first in the reserve price setting and then in the BIN price setting.

3.3.1 Reserve Price Setting

For the reserve price setting in which bidders are loss averse, the experiments cover the case where prior knowledge is correct, two cases where prior knowledge is incorrect, and a non-stationary case.

3.3.1.1 Correct Prior Knowledge

The first experiment considers the best case scenario, in which the seller's prior knowledge of bidder behavior is perfect. In other words, the same distribu-

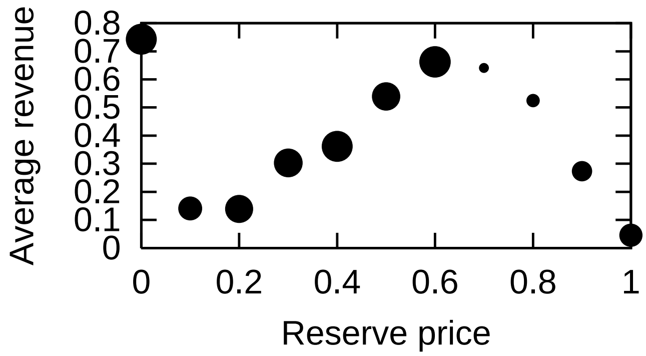

Figure 3.2: Metalearned parameters. Circles represent prior experience. Each circle's height represents the initial value of avg at its position, while the size represents the initial value of $count$. $\tau_{start} = .0081$, $\tau_{end} = .0013$, kernel width = .138

tion over bidder populations is used for both the simulated populations and the encountered populations, meaning that the metalearned regression and Bayesian approaches should both perform well.

During the metalearning step, we began the search process using the same values used for optimistic initialization in Chapter 2: 0.6 for each avg_i and 1 for each $count_i$. τ_{start} and τ_{end} were set to 0.1 and 0.01, respectively, and the kernel width was 0.1. Increasing k beyond 11 did not appear to provide any benefit, and so results are presented for $k = 11$. The learned parameters are presented in Figure 3.2. Prior experience is displayed visually by plotting a circle for each avg_i with area proportional to $count_i$. The largest circle represents a $count$ value of about 4.2. This prior experience appears reasonable given Figure 3.1. The values of $count$ are similar in most cases, but the values of avg are higher in the more promising regions, encouraging initial exploration of those auction parameters. The reasons for such small $count$ values at 0.7 and 0.8 are not immediately clear. The search results appear stable in that repeated runs result in parameters that are fairly similar and provide nearly identical expected revenue per episode, and modest changes to the initial parameters do not significantly affect the search results.

Adaptive algorithm	Avg. revenue
Bayesian approach	0.407
metalearned regression	0.405
optimistic regression	0.385
best fixed reserve price (0.54)	0.367

Table 3.2: Average revenue per auction for each adaptive algorithm over 10,000 populations. Differences are statistically significant at the 99% confidence level according to paired t-tests.

As in Chapter 2, we found the average revenue per adaptive algorithm over 10,000 randomly sampled bidder populations. These average revenues are shown in Table 3.2, while a plot of the average revenue for each auction over an entire episode is shown in Figure 3.3. The difference observed between each pair of methods is statistically significant at the 99% confidence level according to paired t-tests comparing results for the same bidder population. From these results it is clear that the metalearning process is highly effective at identifying effective learning parameters. Metalearned regression learns much faster initially than optimistic regression. In fact, the average revenue reached on the 100th auction by metalearned regression is not reached until after at least 500 auctions by optimistic regression. Thus, the metalearned parameters are effective at focusing initial exploration, providing sufficient prior experience to permit a higher initial degree of exploitation, or both.

The Bayesian approach produces the highest average revenue per auction in this experiment by a small but statistically significant amount. This result is not surprising considering that this approach relies on much stronger assumptions about bidder behavior than the other approaches. It is interesting to note the slight decrease in the performance of the Bayesian approach after around 100 auctions. It may be the case that Bayesian approach converges to slightly suboptimal auction parameters because we consider a discretized space of bidder populations.

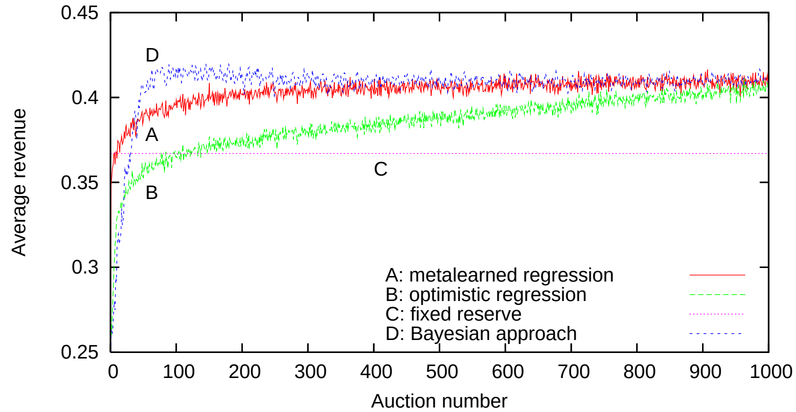

Figure 3.3: Average revenue per auction over the course of an episode for each adaptive algorithm when setting reserve prices.

3.3.1.2 Incorrect Prior Knowledge - Random Bidders

Although adaptive auction mechanisms have been proposed as a means of dealing with unknown bidder behavior, in the previous experiment we evaluated adaptive algorithms on the same distribution of bidder populations used in simulation. The next two experiments investigate the effects when the encountered bidder population differs from these simulated populations.

We first consider the case where the bidders' strategy differs from the expected strategy. As an alternative to the equilibrium bidding strategy described in Section 2.3.1, consider a strategy in which a bidder simply chooses a value uniformly at random from the range [0, 1.2] and bids this value regardless of any other factors. This behavior is clearly different from that present in any of the possible bidder populations considered previously. If bidders using such a strategy were to participate in auctions based on an adaptive mechanism developed using the simulated bidder populations, the adaptive mechanism might not produce the results expected.

To explore the effect of unexpected bidder behavior, we consider populations in which bidders follow this alternate strategy with probability p, and follow the previous equilibrium strategy with probability $1-p$. We consider values of p ranging from 0 to 1 in order to measure the effect of increasingly unexpected behavior.

Figure 3.4 shows the results for all values of p. Because the alternate bidding strategy results in higher bids on average, average revenue increases along with p. As the probability of bidders using the alternate strategy increases, the performance of optimistic regression improves relative to the other two methods, and the Bayesian approach is worst in many cases. This result can be explained by the fact that the regression approach with optimistic parameters makes use of no assumptions about bidder behavior, while the Bayesian approach makes very strong assumptions. With a high value of p, the Bayesian approach is never able to identify the optimal reserve price because it is unable to match the population's behavior with one of the populations it believes to be possible. Metalearned regression is eventually able to identify the optimal reserve price, but often much later than optimistic regression, due to the fact that it is concentrating its early exploration in areas that are now less likely to contain the optimal value. Nevertheless, metalearned regression is never the worst approach, and it is also never more than 0.0065 behind the best approach, while the other approaches fall more than twice as far behind the best approach for some values of p.

3.3.1.3 Incorrect Prior Knowledge - Parameter Ranges

Next, we consider the case where the strategy used by the encountered bidders remains the same as before (the equilibrium strategy), but the distributions from which we draw valuations and α values are changed. Recall that these two bidder parameters were previously assumed to fall within the ranges $v \in [0,1]$ and $\alpha \in [1, 2.5]$. We now use the narrower ranges of $v \in [.3, .7]$ and $\alpha \in [1.5, 2]$ when generating an encountered population (but not the simulated populations used in developing the adaptive algorithms). Distributions within these ranges are generated as before (see Section 2.3.1). Such a situation could arise if the seller

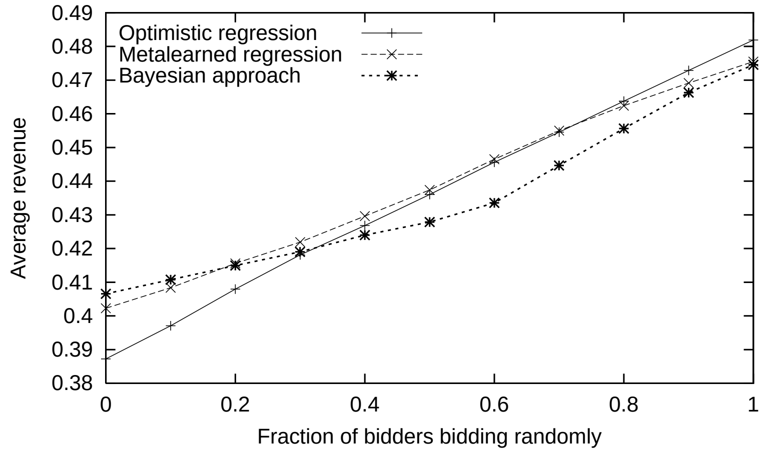

Figure 3.4: Results when some fraction of bidders choose a bid randomly.

was unsure of the actual ranges of v and α and intentionally used conservative estimates of these ranges when developing the adaptive algorithm.

Table 3.3 shows the results. In this case, the performance of metalearned regression is again much better then optimistic regression, suggesting that the behavior of bidders remains sufficiently similar to make metalearning useful in this scenario. The performance of the Bayesian approach suffers even more than in the previous experiment, however. Again, this approach is often unable to match the encountered population's behavior with one of the populations it believes to be possible. This problem appears to be particularly severe when the optimal reserve price is near zero–in many such cases the population's behavior most closely resembles the behavior of populations for which a much higher reserve price is optimal, and so the Bayesian approach consistently tries a suboptimal reserve price.

The results of the last two experiments show that there is a clear tradeoff in the use of prior knowledge but suggest that this tradeoff can be managed effectively. An approach that relies on specific knowledge of bidders' behaviors to adapt is able to identify effective auction parameter values more quickly than a purely data-driven approach when the seller's prior knowledge is correct, but it may seriously

Adaptive method	Avg. revenue
best fixed reserve price (0.54)	0.335
Bayesian approach	0.414
optimistic regression	0.575
metalearned regression	0.593

Table 3.3: Average revenue per auction for each adaptive algorithm when bidder parameters are drawn from modified distributions. Differences are statistically significant at the 99% confidence level according to paired t-tests.

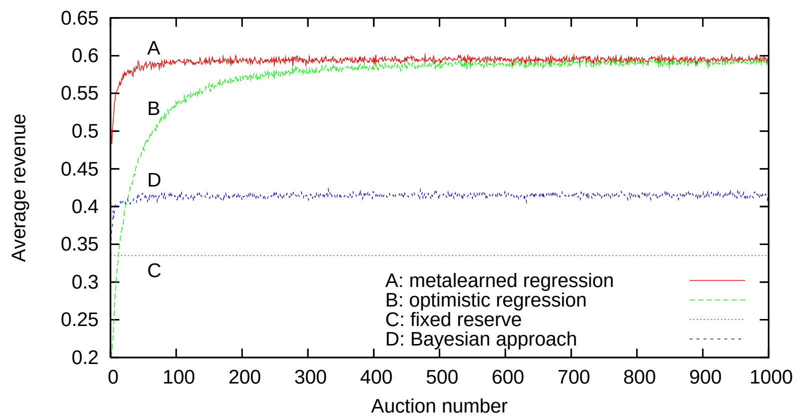

Figure 3.5: Average revenue per auction over the course of an episode for each adaptive algorithm with an unexpected population parameter distribution.

suffer if this knowledge is incorrect. On the other hand, an approach that learns only from experience with the encountered population is unable to exploit prior knowledge to reduce costly exploration; however, this approach can eventually learn effective auction parameter values regardless of the correctness of this knowledge. Overall, metalearned regression appears to strike a useful balance between using prior knowledge and learning directly from experience.

3.3.1.4 Non-stationary Bidders

In the previous experiments, the behavior of the bidder population remained constant over the entire episode. Such consistency is unlikely in the real world.

Possible causes of shifts in bidder behavior include changes in the preferences of bidders and changes in the strategies employed by bidders. Bidder preferences may change gradually, such as when valuations decrease over time for outdated products, or they may change quickly, such as when a new type of bidder suddenly enters the marketplace. Changes in bidder strategies may be the result of the bidders themselves adapting by learning from past experiences.

This section presents modifications to metalearned regression to cope with a gradually changing bidder population. Dealing with more sudden changes would likely require more extensive modifications, such as attempting to determine precisely when a change has occurred, and is beyond the scope of this dissertation. Recall that the behavior of our simulated bidder population is controlled by four parameters: the mean and variance of the valuations, and the mean and variance of α values. We now allow each of these parameters to vary according to a random walk: after each auction, each parameter is either increased or decreased by one percent of the total range for that parameter, within the bounds specified in Section 2.3.1. Bidder populations are initialized as before.

Under this scenario, the probability of a particular set of population parameter values occurring during an auction is essentially the same as before, and therefore the optimal fixed reserve remains 0.54, with an average revenue of 0.367. Using metalearned regression with the previously metalearned parameters results in an average revenue of 0.379, still a significant improvement over the fixed reserve but well below the 0.405 obtained previously.

One way of dealing with a changing bidder population is to weight recent experience more heavily than older (and possibly more inaccurate) experience. We implement this modification by choosing a decay rate γ by which the weight of all past experience (including the prior experience) is multiplied at each step. Recall that LWQR assigns a weight to each past experience, e^{d/w^2}, where d is the distance to the auction parameter value for which revenue is to be predicted. To discount older outcomes, the weight assigned to an auction outcome which occurred at time l is now given by $e^{d/w^2}\gamma^{t-l}$, where t is the current auction number. Because observations with lower weights will have less impact on LWQR's prediction, these weights cause older experiences to have a lower effect on the seller's estimations of expected revenue When using a decay rate of 0.99, chosen manually after limited experimentation, the average revenue per auction increases to 0.383, a modest improvement over the previous 0.379.

Another possible means of improving performance with a non-stationary population is to increase the temperature τ used in selecting reserve prices for each auction. Recall that τ controls the degree of exploration when using softmax action selection, and that in the previous experiments τ was gradually decreased over time to encourage increased exploitation once reasonable estimates of auction outcomes had been learned. Because these estimates may become inaccurate as the bidder population changes, maintaining a higher degree of exploration by decreasing τ more slowly might be a way to improve performance of the adaptive mechanism. For this particular scenario, however, experiments show that the average revenue remains almost unchanged when τ is decreased more slowly (with or without the decay of experience), suggesting that the gains from increased exploration are offset by the losses from reduced exploitation. (With the stationary bidder population considered previously, decreasing τ more slowly results in a significant reduction in average revenue, so it is not simply the case that τ is unimportant.)

If the seller is aware of the specific way in which the population can change, then the seller can make use of this information by performing the metalearning step using the non-stationary population in simulation. Although the learned parameters that result (prior experience, temperatures, kernel width, and decay rate) are fairly similar to those used previously–the most notable changes are a decrease in the decay rate to 0.985 and an increase in kernel width to 0.154–the average

Adaptive method	Avg. revenue
best fixed reserve price (0.54)	0.367
previous metalearned parameters	0.379
previous metalearned parameters, experience decay	0.383
new metalearned parameters, exp. decay	0.389
new metalearned parameters, no initial exp. decay	0.394

Table 3.4: Average revenue per auction for each adaptive method with a non-stationary population. Differences are statistically significant at the 99% confidence level according to paired t-tests.

revenue increases to 0.389.

The method of decaying past experience described above includes the decay of the prior experience (the values stored before actual experience is obtained). The prior experience could instead be handled separately and have its weight remain unchanged. For reserve prices with little recent experience, the predicted revenue would then be based largely on the initial values, potentially encouraging periodic re-exploration of reserve prices previously determined to be suboptimal. For a non-stationary population, such re-exploration could possibly be beneficial. When using this approach and applying metalearning with full knowledge of how the population changes, the average revenue obtained reaches 0.394.

The results of this section, summarized in Table 3.4, demonstrate that metalearned regression can be applied in situations in which bidder populations vary over time, at least when such change is gradual. When knowledge of how the population might change is available, this knowledge can be usefully incorporated into the process of metalearning optimal learning parameters.

3.3.2 BIN Price Setting

The remaining experiments confirm the effectiveness of metalearned regression in the BIN price setting with time-sensitive bidders with both correct and incorrect prior knowledge.

3.3.2.1 Correct Prior Knowledge

As before, the first experiment in this setting considers the best case scenario, in which the seller's prior knowledge of bidder behavior is perfect. Using the distribution over bidder populations described in Section 2.3.2 and performing metalearning using the optimistic learning parameters as the starting point of the search ($\tau_{start} = 0.01$, $\tau_{end} = 0.001$, each $count_i = 1$, each $avg_i = 1$, and kernel width 0.1) resulted in the learning parameters shown in Figure 3.6. Again, the height of each circle represents the profit, and the area represents the weight–the largest circle has weight 3.11, and the smallest 0.02. Increasing k (the number of auction parameter values with prior experience) beyond 11 did not appear to provide any benefit.

The prior experience shown in shown in Figure 3.6 makes sense given the nature of BIN prices. When the BIN price is set too low, there can be a large loss in profit due to the fact that the BIN price acts as an upper bound on revenue, but setting the BIN price too high is essentially the same as having no BIN price, which is only moderately harmful. Therefore, a good exploration policy is to begin with a high BIN price and then reduce it until the optimal price is found, and this is essentially the exploration policy induced by the prior experience. While it is likely that a human designing these auctions would have observed this fact and been able to implement such a strategy by hand, it is reassuring that the metalearning process was able to learn this behavior, and it is possible that in other auction settings there would be other factors that would be picked up by the metalearning process but not by a human auction designer.

Table 3.5 shows the average profit per auction (revenue - seller's time cost) for each algorithm over 10,000 episodes, and Figure 3.7 plots the average over the course of an episode. Metalearned regression performs as well as the best fixed BIN price from the start of each episode and identifies a nearly optimal BIN price in

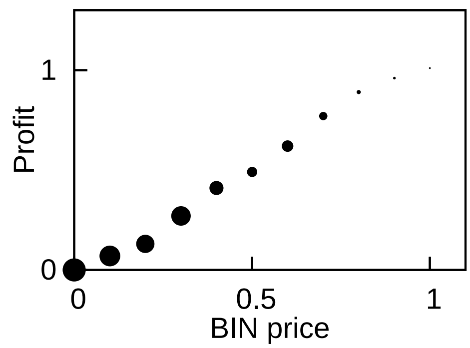

Figure 3.6: Metalearned parameters. Circles represent prior experience. Each circle's height represents the initial value of *avg* at its position, while the size represents the initial value of *count*. $\tau_{start} = .013$, $\tau_{end} = .002$

Adaptive method	Avg. profit
best fixed BIN price (0.73)	0.4884
optimistic regression	0.4711
metalearned regression	0.5254

Table 3.5: Average profit per auction for each adaptive algorithm for the BIN price setting. Differences are statistically significant at the 99% confidence level according to paired t-tests.

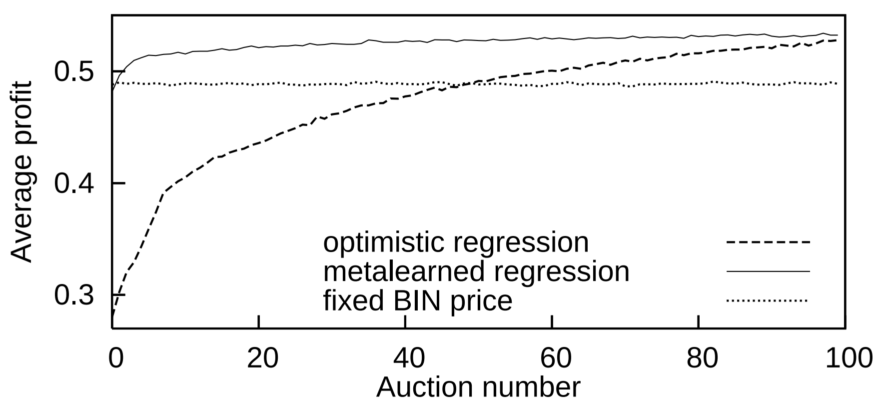

Figure 3.7: Average profit per auction over the course of an episode for each adaptive algorithm when setting BIN prices.

only a few auctions.

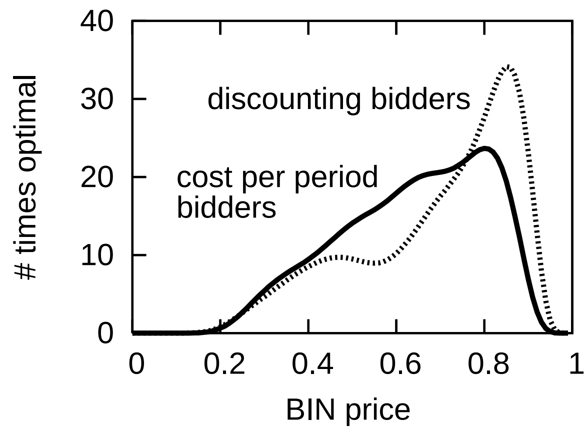

Figure 3.8: Number of times each BIN price was optimal out of 1000 populations considered for the two kinds of bidders.

3.3.2.2 Incorrect Prior Knowledge - Exponential Discounting

The next experiment considers the case that the seller's prior knowledge includes incorrect assumptions about the nature of bidders' time sensitivity. Instead of having a cost per period, the encountered bidders discount their surplus exponentially. Thus, a bidder with discount factor γ and surplus s that participated for n periods would view its reward as $\gamma^n s$. Bidder populations are generated just as before, except that instead of choosing a distribution over costs per period for each population, we choose a distribution over discount factors by taking a Gaussian with a mean chosen uniformly from [0.992, 0.998] and a standard deviation chosen uniformly from [0.001, 0.004], and then normalizing the distribution so that the portion restricted to the range [0.99, 1] represents a PDF. The bidding strategy is learned as before using competitive coevolution, and the bidder behavior that results appears somewhat different from the behavior of the cost per period bidders. Figure 3.8 shows that the distribution over optimal BIN prices is slightly changed.

We used the same optimistic and metalearned learning parameters as before and ran simulations with 10,000 populations of exponential discounting bidders.

Adaptive method	Avg. profit
best fixed BIN price (0.73)	0.5093
optimistic regression	0.5239
metalearned regression	0.5504

Table 3.6: Average profit per auction for each adaptive algorithm for the BIN price setting with exponential discounting bidders. Differences are statistically significant at the 99% confidence level according to paired t-tests.

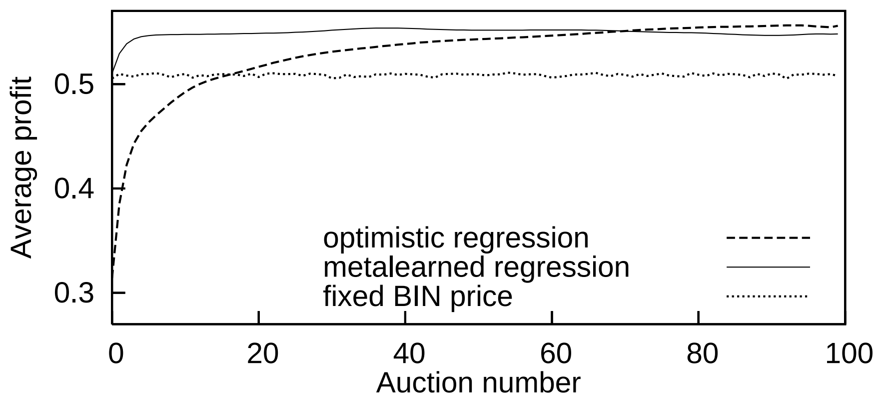

Figure 3.9: Average profit per auction over the course of an episode for each adaptive algorithm with exponential discounters.

Table 3.6 shows the average profit per auction for each algorithm, and Figure 3.9 plots the average over the course of an episode. Again, the metalearned regression resulted in the highest overall profit. It appears that the exploration strategy induced by the prior experience remains effective when bidders are exponential discounters. Still, optimistic regression ultimately found a better BIN price, on average, than metalearned regression. It appears that metalearned regression reduced its exploration early and settled on a particular BIN price, and while this prevented the approach from finding a better BIN price, the lack of costly exploration still helped metalearned regression outperform optimistic regression.

3.3.2.3 Best-response Bidders

Up to this point, we have assumed that the behavior of bidders is independent of the adaptive strategy used by the seller. That is, bidders respond to the information available about the particular auction in which they are bidding, but are unaware that the auction parameter values chosen by the seller for future auctions may be dependent on the bidders' actions in the current auction. In cases in which bidders may remain active for several consecutive auctions before winning or giving up, and especially in cases where bidders experience recurring demand and may return to participate in future auctions, bidders may have an incentive to act in ways designed to influence the future actions of the seller. (For instance, by refusing to accept a high BIN price in early auctions, bidders could induce the seller to offer lower BIN prices in later auctions.) Bidders could gradually learn such responsive strategies through their interactions with the seller, or could even employ strategies specifically designed to be used with the seller if they knew the seller's adaptive strategy in advance. We explore the latter case, as it is essentially the worst case scenario for the seller.

We can use the coevolutionary process introduced in this chapter to develop this type of responsive bidder strategy. Instead of using the random seller strategy as before when evaluating bidder strategies, we use the regression adaptive algorithm with metalearned parameters. In addition to the information previously available to bidders, bidders are now also told which auction number (between 1 and 100) is currently running. Each bidder strategy is used several times throughout an episode, allowing a strategy to be rewarded for causing the seller to choose favorable auction parameter values in later auctions. Bidders are modeled as having a cost per period as before.

The results of evaluating our adaptive algorithms with the final bidder strategy evolved in this way are similar to the previous results for the case of exponential

Adaptive method	Avg. profit
best fixed BIN price (0.73)	0.3967
optimistic regression	0.3912
metalearned regression	0.4135

Table 3.7: Average profit per auction for each adaptive algorithm for the BIN price setting with best-response bidders. Differences are statistically significant at the 99% confidence level according to paired t-tests.

discounting bidders: although the encountered bidder population behaves somewhat differently than expected, metalearned regression is still effective at quickly learning profitable BIN prices. Table 3.7 shows the average profit per auction for each algorithm, and Figure 3.10 shows the average over the course of an episode. One interesting observation is that the average profit per auction is about 0.1 less than before for all three algorithms. It appears that bidders tend to drop out of the auctions earlier, reducing competition. One possible explanation for this result is that when the random seller strategy is used during evolution, bidders learn to hang around in the hopes that they can take advantage of an abnormally low BIN price, but such opportunities are rarer when a more intelligent seller strategy is used. In any case, these results are reassuring in that, at least for this setting, they suggest that adaptive auction mechanisms do not open the seller to exploitation by the bidders. These results also serve as further evidence that metalearned regression can still be effective when bidders behave in an unexpected fashion.

3.4 Summary

This chapter introduced two extensions to the regression adaptive algorithm described in Chapter 2 that can be used when the seller has sufficient prior knowledge about bidders that it can simulate a range of plausible bidder behaviors: using metalearning to find the optimal learning parameters in simulation, and a Bayesian approach to identifying which simulated population the encountered bidder popu-

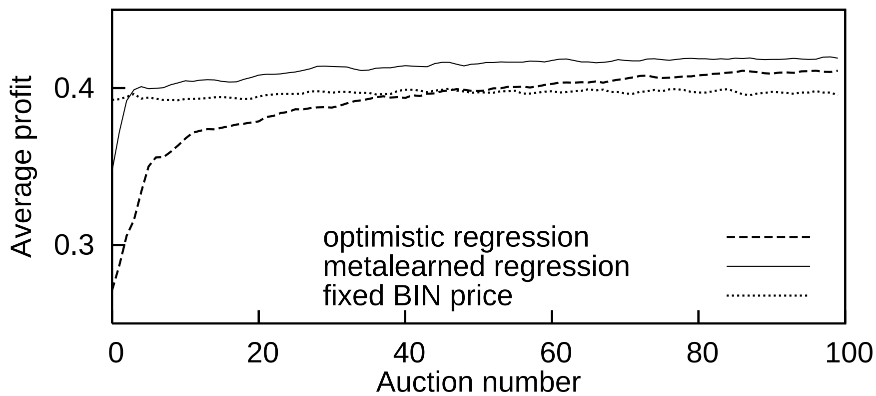

Figure 3.10: Average profit per auction over the course of an episode for each adaptive algorithm with learning bidders.

lation is most similar to. The Bayesian approach is able to learn effective auction parameters very quickly when the encountered population does come from the space of simulated populations. When bidders in the encountered population behave in unexpected ways, however, this approach may be unable to adapt.

The metalearning regression approach, on the other hand, is robust under a variety of circumstances. When the encountered population is similar to the simulated populations, metalearning regression is able to perform nearly as well as the Bayesian approach. When confronted with unexpected bidder behavior, metalearning regression is still able to find effective auction parameters and continues to outperform optimistic regression in most cases. In addition, this chapter has shown how metalearning regression can be successfully applied in the case of non-stationary bidders, and there is evidence that metalearning regression remains effective even when bidders know that an adaptive auction mechanism will be used.

Chapter 4

The TacTex Agent for Supply Chain Management

I now move on to a new setting, the Trading Agent Competition (TAC). Within this setting I consider two scenarios: Supply Chain Management (SCM) and Ad Auctions (AA). Discussion of this setting is spread over six chapters. In the first four chapters, I describe the two scenarios and present our winning agent in each. The purpose of these chapters is to place the learning problem in the context of a full agent and motivate the need for adapting to opponent behavior.

This chapter covers the TacTex agent for TAC SCM. I give a summary of the scenario, describe the design of the TacTex agent, and then present experimental results highlighting the importance of various agent components. This chapter fulfills Contribution 1 for TAC SCM, as outlined in Section 1.2. In the following chapter, I will address the use of machine learning to further improve performance.

4.1 TAC SCM Overview

In today's industrial world, supply chains are ubiquitous in the manufacturing of many complex products. Traditionally, supply chains have been created through the interactions of human representatives of the various companies involved. However, recent advances in autonomous agent technologies have sparked an interest, both in academia and in industry, in automating the process [62] [86] [26]. Creating a fully autonomous agent for supply chain management is difficult due to the large number of tasks such an agent must perform. In general, the agent must procure resources for, manage the assembly of, and negotiate the sale of a completed product. To perform these tasks intelligently, the agent must be able to plan in the face of uncertainty, schedule the optimal use of its resources, and adapt to changing market conditions.

TAC SCM provides a unique testbed for studying and prototyping supply chain management agents by providing a competitive environment in which independently created agents can be tested against each other over the course of many simulations in an open academic setting. In a TAC SCM game, each agent acts as an independent computer manufacturer in a simulated economy. The agent must procure components such as CPUs and memory, decide what types of computers to manufacture from these components as constrained by its factory resources, bid for sales contracts with customers, and decide which computers to deliver to whom and by when.

4.2 TAC SCM Specification

I now present a summary of the TAC SCM scenario. Full details are available in the official specification document [28].

In a TAC SCM game, six agents act as computer manufacturers in a simulated economy that is managed by a game server. The length of a game is 220 simulated days, with each day lasting 15 seconds of real time. At the beginning of each day, agents receive messages from the game server with information concerning the state of the game, such as the customer requests for quotes (RFQs) for that day, and agents have until the end of the day to send messages to the server indicating their actions for that day, such as making offers to customers. The game can be divided into three parts: i) component procurement, ii) computer sales, and iii) production and delivery as expanded on below and illustrated in Figure 4.1.

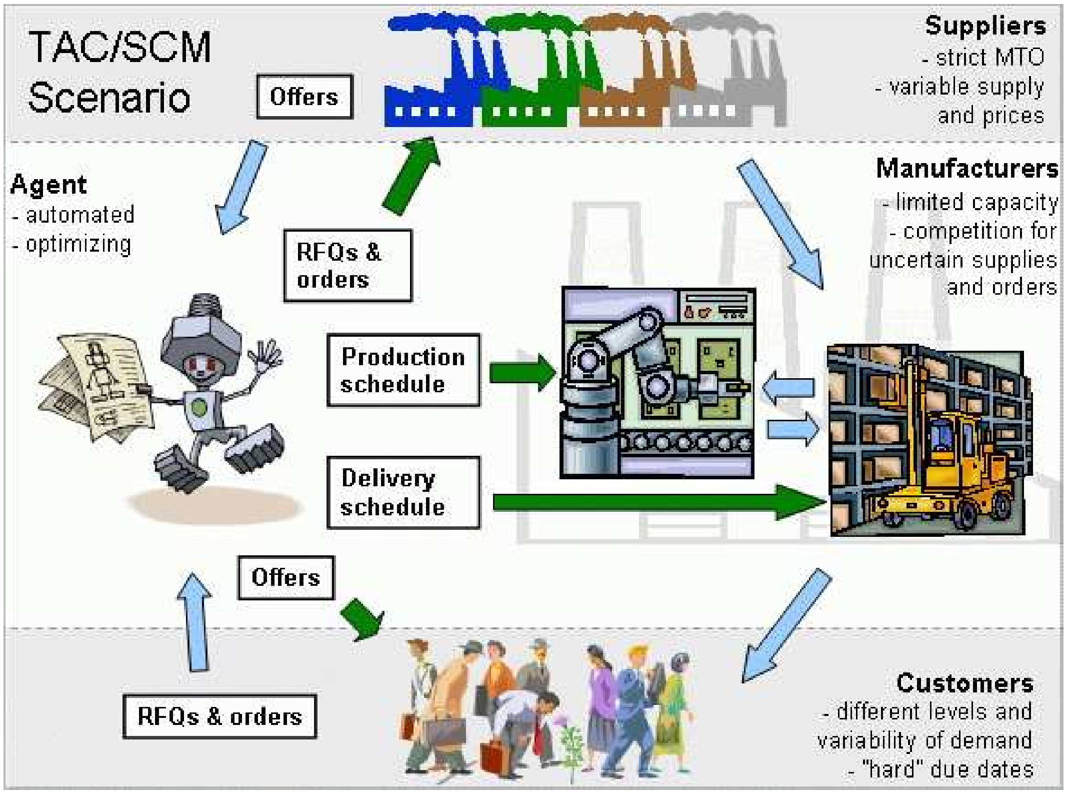

Figure 4.1: The TAC SCM Scenario [28].

4.2.1 Component Procurement

The computers are made from four components: CPUs, motherboards, memory, and hard drives, each of which come in multiple varieties. From these components, 16 different computer configurations can be made. Each component has a *base price* that is used as a reference point by suppliers making offers.

Agents wanting to purchase components send requests for quotes (RFQs) to suppliers indicating the *type and quantity* of components desired, the *date* on which they should be delivered, and a *reserve price* stating the maximum amount the agent is willing to pay. Agents are limited to sending at most 5 RFQs per component per supplier per day. Suppliers respond to RFQs the next day by offering a price for the requested components if the request can be satisfied. Agents may then accept or reject the offers.

Suppliers have a *limited capacity* for producing components, and this capacity varies throughout the game according to a random walk. Suppliers base their prices offered in response to RFQs on the fraction of their capacity that is currently free. When determining prices for RFQs for a particular component, a supplier simulates scheduling the production of all components currently ordered plus those components requested in the RFQs as late as possible. From the production schedule, the supplier can determine the remaining free capacity between the current day and any future day. The price offered in response to an RFQ is equal to the base price of the component discounted by an amount proportional to the fraction of the supplier's capacity free before the due date. Agents may send zero-quantity RFQs to serve as price probes. Due to the nature of the supplier pricing model, it is possible for prices to be as low when components are requested at the last minute as when they are requested well in advance. Agents thus face an interesting *tradeoff*: they may either commit to ordering while knowledge of future customer demand is still limited (see below), or wait to order and risk being unable to purchase needed components.

To prevent agents from driving up prices by sending RFQs with no intention of buying, each supplier keeps track of a *reputation* rating for each agent that represents the fraction of offered components that have been accepted by the agent. If this reputation falls below a minimum acceptable purchase ratio (75% for CPU suppliers, and 45% for others), then the prices and availability of components are affected for that agent. Agents must therefore plan component purchases carefully, sending RFQs only when they believe it is likely that they will accept the offers received.

4.2.2 Computer Sales

Customers wishing to buy computers send the agents RFQs consisting of the *type and quantity* of computer desired, the *due date*, a *reserve price* indicating the maximum amount the customer is willing to pay per computer, and a *penalty* that must be paid for each day the delivery is late. Agents respond to the RFQs by bidding in a first-price auction: the agent offering the lowest price on each RFQ wins the order. Agents are unable to see the prices offered by other agents or even the winning prices, but they do receive a report each day indicating the highest and lowest price at which each type of computer sold on the previous day.

Each RFQ is for between 1 and 20 computers, with due dates ranging from 3 to 12 days in the future, and reserve prices ranging from 75% to 125% of the base price of the requested computer type. (The base price of a computer is equal to the sum of the base prices of its parts.)

The number of RFQs sent by customers each day depends on the level of customer demand, which fluctuates throughout the game. Demand is broken into three segments, each containing about one third of the 16 computer types: high, mid, and low range. Each range has its own level of demand. The total number of RFQs per day ranges between roughly 80 and 320, all of which can be bid upon by all six agents. It is *possible for demand levels to change rapidly*, limiting the ability of agents to plan for the future with confidence.

4.2.3 Production and Delivery

Each agent manages a factory where computers are assembled. Factory operation is constrained by both the components in inventory and assembly cycles. Factories are limited to producing roughly 360 computers per day (depending on their types). Each day an agent must send a production schedule and a delivery schedule to the server indicating its actions for the next day. The production schedule specifies how many of each computer will be assembled by the factory, while the delivery schedule indicates which customer orders will be filled from the completed computers in inventory. Agents are required to pay a small daily storage fee for all components in inventory at the factory. This cost is sufficiently high to discourage agents from holding large inventories of components for long periods.

4.3 Overview of TacTex

Given the detail and complexity of the TAC SCM scenario, creating an effective agent requires the development of tightly coupled modules for interacting with suppliers, customers, and the factory. The fact that each day's decisions must be made in less than 15 seconds constrains the set of possible approaches.

TacTex is a fully implemented agent that operates within the TAC SCM scenario. In this section, I present a high-level overview of the agent. TacTex was developed over a number of years, with incremental improvements each year. The basic agent design described in this section was first present in the 2005 agent (the first version of TacTex to win an annual TAC SCM tournament) and has remained unchanged since then. In the remaining sections of this chapter, I describe the components that were present in the 2005 agent (Sections 4.4 and 4.5), which rely on a number of heuristics to make predictions. The description of the learning based predictors that were added in later years is left to the following chapter.

4.3.1 Agent Components

Figure 4.2 illustrates the basic components of TacTex and their interaction. There are five basic tasks a TAC SCM agent must perform:

1. Sending RFQs to suppliers to request components;

2. Deciding which offers from suppliers to accept;

3. Bidding on RFQs from customers requesting computers;
4. Sending the daily production schedule to the factory;
5. Delivering completed computers.

We assign the first two tasks to a *Supply Manager* module, and the last three to a *Demand Manager* module. The Supply Manager handles all planning related to component inventories and purchases, and requires no information about computer production except for a projection of future component use, which is provided by the Demand Manager. The Demand Manager, in turn, handles all planning related to computer sales and production. The only information about components required by the Demand Manager is a projection of the current inventory and future component deliveries, along with an estimated replacement cost for each component used. This information is provided by the Supply Manager.

We view the tasks to be performed by these two managers as optimization tasks: the Supply Manager tries to minimize the cost of obtaining the components required by the Demand Manager, while the Demand Manager seeks to maximize the profits from computer sales subject to the information provided by the Supply Manager. In order to perform these tasks, the two managers need to be able to make predictions about the results of their actions and the future of the economy. TacTex uses three predictive models to assist the managers with these predictions: a predictive *Supplier Model*, a predictive *Demand Model*, and an *Offer Acceptance Predictor*.

The Supplier Model keeps track of all information available about each supplier, such as TacTex's outstanding orders and the prices that have been offered in response to RFQs. Using this information, the Supplier Model can assist the Supply Manager by making predictions concerning future component availability and prices.

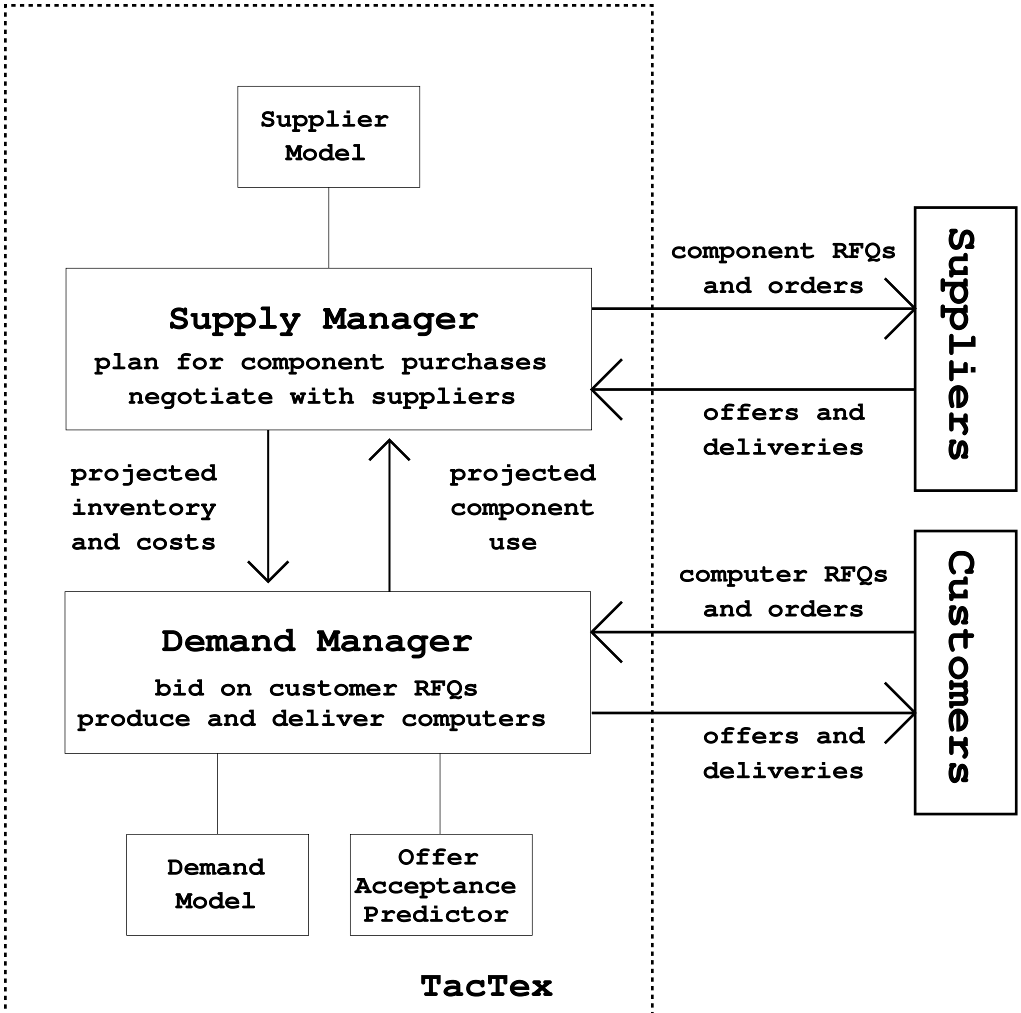

Figure 4.2: An overview of the main agent components

The Demand Model tracks the customer demand in each of the three market segments, and tries to estimate the underlying demand parameters in each segment. With these estimates, it is possible to predict the number of RFQs that will be received on any future day. The Demand Manager can then use these predictions to plan for future production.

- Record information received from the server and update prediction modules
- The Supply Manager takes the supplier offers as input and performs the following:
  - decide which offers to accept
  - update projected future inventory
  - update replacement costs
- The Demand Manager takes customer RFQs, current orders, projected inventory, and replacement costs as input and performs the following:
  - predict future customer demand using the Demand Model
  - use the Offer Acceptance Predictor to generate acceptance functions for RFQs
  - schedule production several days into the future
  - extract the current day's production, delivery, and bids from the schedule
  - update projected future component use
- The Supply Manager takes the projected future component use as input and performs the following:
  - determine the future deliveries needed to maintain a threshold inventory
  - use the Supplier Model to predict future component prices
  - decide what RFQs need to be sent on the current day

Table 4.1: Overview of the steps taken each day by TacTex.

When deciding what bids to make in response to customer RFQs, the Demand Manager needs to be able to estimate the probability of a particular bid being accepted (which depends on the bidding behavior of the other agents). This prediction is handled by the Offer Acceptance Predictor. Based on past bidding results, the Offer Acceptance Predictor produces a function for each RFQ that maps bid prices to the predicted probability of winning the order.

The steps taken each day by TacTex as it performs the five tasks described previously are presented in Table 4.1.

4.4 The Demand Manager

The Demand Manager handles all computation related to computer sales and production. This section describes the Demand Manager, along with the Demand Model and the Offer Acceptance Predictor upon which it relies. The Offer Acceptance Predictor described here was present in the 2005 agent; Section 5.1 of the following chapter describes an improved version used in later years.

4.4.1 Demand Model

When planning for future computer production, the Demand Manager needs to be able to make predictions about future demand in each market segment. For example, if more RFQs are expected for high range than low range computers, the planned production should reflect this fact. The Demand Model is responsible for making these predictions.

In order to explain its operation, further detail is required about the customer demand model. The state of each demand segment (high, mid, and low range computers) is represented by parameters Q_d and τ_d (both of which are internal to the game server). Q_d represents the expected number of RFQs on day d, and τ_d is the trend in demand (increasing or decreasing) on day d. The actual number of RFQs is generated randomly from a Poisson distribution with Q_d as its mean. The next day's demand, Q_{d+1}, is set to $Q_d\tau_d$, and τ_{d+1} is determined from τ_d according to a random walk.

To predict future demand, the Demand Manager estimates the values of Q_d and τ_d for each segment using a grid-based filtering approach first used by the agent DeepMaize in 2003 [57]. Basically, this is a Bayesian approach that involves maintaining a probability distribution over (Q_d, τ_d) pairs for each segment. The number of RFQs received each day from the segment represents information that can be used to update this distribution, and the distribution over (Q_{d+1}, τ_{d+1}) pairs can then be generated based on the game's demand model. By repeating this last step, the expected value of Q_i can be determined for any future day i and used as the number of RFQs predicted on that day. Full details of the approach are available in [57].

4.4.2 Offer Acceptance Predictor

In order to bid on customer RFQs, the Demand Manager needs to be able to predict the orders that will result from the offers it makes. A simple method of prediction would be to estimate the winning price for each RFQ, and assume that any bid below this price would result in an order. Alternatively, for each RFQ the probability of winning the order could be estimated as a function of the current bid. This latter approach is the one implemented by the Offer Acceptance Predictor. For each customer RFQ received, the Offer Acceptance Predictor generates a function mapping the possible bid prices to the probability of acceptance. (The function can thus be viewed as a cumulative distribution function.) The approach used in the 2005 agent involves two components: a linear heuristic for generating a function, and an adaptive means of revising the heuristic's predictions.

The linear heuristic is based on one used by the agent Botticelli in 2003 [10] whereby the CDF generated for each RFQ depends only on the type of computer requested and is generated using linear regression on six data points. Specifically, for each of the past five days, the average price bid by TacTex for the given type of computer is determined, along with the fraction of offers accepted on that day. Each pair results in one data point, and the sixth point represents the highest winning price reported for the given type of computer on the previous day along with an acceptance rate of zero. (Unlike [11], we do not add a point representing the lowest reported winning price, because it only takes a single agent significantly underbidding on a single RFQ to make this point a poor representative of the true function.) These points are fit using least squares linear regression to generate a linear function that will be used for all RFQs requesting the given computer type.

The linear function is modified using values we call *day factors*, which are designed to measure the effect of the due date on offer acceptance. The due dates for RFQs range from 3 to 12 days in the future, and a separate day factor is learned for each day in this range. Each day factor is set to the ratio of actual orders received to orders expected based on the linear heuristic, for all recent offers made. When an offer is made on an RFQ, the Offer Acceptance Predictor computes the probability of an order by multiplying the initial prediction by the corresponding day factor. The day factors therefore serve both as a means of gauging the impact of due dates on computer prices and as a mechanism for ensuring that the number of orders received is roughly the number expected.

4.4.3 Demand Manager

The Demand Manager is responsible for bidding on customer RFQs, producing computers, and delivering them to customers. All three tasks can be performed using the same production scheduling algorithm. As these tasks compete for the same resources (components, completed computers, and factory cycles), the Demand Manager begins by planning to satisfy existing orders, and then uses the remaining resources in planning for RFQs. The latest possible due date for an RFQ received on the current day is 12 days in the future, meaning the production schedule for the needed computers must be sent within the next 10 days. The Demand Manager thus always plans for the next 10 days of production. Each day, the Demand Manager i) schedules production of existing orders, ii) schedules production of predicted future orders, and then iii) extracts the next day's production and delivery schedule from the result. The production scheduling algorithm, these three steps, and the means of predicting production beyond 10 days are described in the following sections.

4.4.3.1 Production Scheduling Algorithm

The goal of the production scheduler is to to take a set of orders and determine the 10-day production schedule that maximizes profit, subject to the available resources. The resources provided are:

- A fixed number of factory cycles per day;
- The components in inventory;
- The components projected to be delivered; and
- Completed computers in inventory.

The profit for each order is equal to its price (if it could be delivered) minus any penalties for late delivery and the replacement costs for the components involved as specified by the Supply Manager.

The scheduling algorithm used by the Demand Manager is a greedy algorithm that attempts to produce each order as late as possible. Orders are sorted by profit, and the scheduler tries to produce each order using cycles and components from the latest possible dates. If any part of the order cannot be produced, the needed computers will be taken from the existing inventory of completed computers, if possible. The purpose of scheduling production as late as possible is to preserve resources that might be needed by orders with earlier due dates. A record is kept of what production took place on each day and how each order was filled.

It should be noted that the scheduling problem at hand lends itself to the use of linear programming to determine an optimal solution. We initially experimented with this approach, using a linear program similar to one designed for a slightly simplified scenario by [10]. However, due to the game's time constraints (15s allowed per simulated day), the need to use the scheduler multiple times per day (and in a modified fashion for bidding on customer RFQs, as described below), and the fact that the greedy approach is nearly optimal (observed in our own experiments and confirmed by [9]), we chose to use the greedy approach.

4.4.3.2 Handling Existing Orders

The Demand Manager plans for the production of existing orders in two steps. Before starting, the production resources are initialized using the values provided by the Supply Manager. Then the production scheduler is applied to the set of orders due in one day or less. All orders that can be taken from inventory (hopefully all of them to avoid penalties) are scheduled for delivery the next day. The production scheduler is next applied to the remaining orders. No deliveries are scheduled for these orders, because there is no reward for early delivery.

4.4.3.3 Bidding on RFQs and Handling Predicted Orders

The goal of the Demand Manager is now to identify the set of bids in response to customer RFQs that will maximize the expected profit from using the remaining production resources for the next 10 days, and to schedule production of the resulting predicted orders. The profit depends not only on the RFQs being bid on on the current day, but also on RFQs that will be received on later days for computers due during the period. If these future RFQs were ignored when selecting the current day's bids, the Demand Manager might plan to use up all available production resources on the current RFQs, leaving it unable to bid on future RFQs. One way to address this issue would be to restrict the resources available to the agent for production of the computers being bid on (as in [10]). Instead, the Demand Manager generates a predicted set of all RFQs, using the levels of customer demand predicted by the Demand Model, that will be received for computers due during the period, and chooses bids for these RFQs at the same time as the actual RFQs from the current day.

Once the predicted RFQs are generated, the Offer Acceptance Predictor is used to generate an acceptance prediction function for every RFQ, both real and predicted. The Demand Manager then considers the production resources remaining, the set of RFQs, and the set of acceptance prediction functions and simultaneously generates a set of bids on RFQs and a production schedule that produces the expected resulting orders, using the following modification of the greedy scheduler.

If we were considering only a single RFQ and had no resource constraints, the expected profit resulting from a particular bid price would be:

$$Expected\ profit = P(order|price) * (price - cost) \tag{4.1}$$

The optimal bid would be the value that maximized this quantity.

Computing the expected profit from a set of bids when resource constraints are considered is much more difficult, however, because the profit from each RFQ cannot be computed independently. For each possible set of orders in which it is not possible to fill all orders, the profit obtained depends on the agent's production and delivery strategy. For any nontrivial production and delivery strategy, precise calculation of the expected profit would require separate consideration of a number of possible outcomes that is exponential in the number of RFQs. If we were guaranteed that we would be able to fill all orders, we would not have this problem. The expected profit from each RFQ could be computed independently, and we would have:

$$Expected\ profit = \sum_{i\ \epsilon\ all\ RFQs} P(order_i|price_i) * (price_i - cost_i) \tag{4.2}$$

Our bidding heuristic is based on the assumption that the expected number of computers ordered for each RFQ will be the actual number ordered. In other words, we pretend that it is possible to win a part of an order, so that instead of winning an entire order with probability p, we win a fraction p of an order with probability 1. This assumption greatly simplifies the consideration of filling orders, since we now have only one set of orders to consider, while leaving the formulation of expected profit unchanged. As long as it is possible to fill the partial orders, (4.2) will hold, where the probability term now refers to the fraction of the order won. It would appear that this approach could lead to unfilled orders when the agent wins more orders than expected, but in practice, this is not generally a problem. Most of the RFQs being bid on are the predicted RFQs that will be received on future days, and so the agent can modify its future bidding behavior to correct for an unexpectedly high number of orders resulting from the current day's RFQs. TacTex indeed tends to have very few late or missed deliveries using this bidding strategy.

By using this notion of partial orders, we can transform the problem of bid selection into the problem of finding the most profitable set of partial orders that can be filled with the resources available, and we can solve this problem using the greedy production scheduler. All bids are initially set to be just above the reserve price, which means we begin with no orders. The scheduler then chooses an RFQ and an amount by which its bid will be lowered, resulting in an increased partial order for that RFQ. The scheduler simulates filling this increase by scheduling its production as described previously. This process is repeated until no more production is possible or no bid can be reduced without reducing the expected profit.

Because we are working with resource constraints, the goal of the greedy production scheduler at each step is to obtain the largest possible increase in profit using the fewest possible production resources. At each step, the scheduler considers each RFQ and determines the bid reduction that will produce the largest increase in profit per additional computer. The scheduler then selects the RFQ for which this value is the largest. In many cases, however, the most limited resource is

Algorithm 1 `Bidding Heuristic`

Input a set of RFQs and available resources (free cycles, components in inventory, and expected component deliveries) over the desired range of days

- For each RFQ, compute both the probability of winning and the expected profit as a function of price
- Set the bid for each RFQ to be just above the reserve price
- Repeat until no RFQs are left in the list of RFQs to be considered:
  1. For each RFQ, find the bid lower than the current bid that produces the largest increase in profit per additional computer ordered (or per additional cycle required during periods of high factory utilization)
  2. Choose the RFQ and bid that produce the largest increase.
  3. Try to schedule production of the partial order resulting from lowering the bid. If it cannot be scheduled, remove the RFQ from the list.
  4. If the production was scheduled, but no further decrease in the bid will lead to an increase in profit, remove the RFQ from the list.
- Return the final bid for each RFQ.

production cycles, and not components. In such cases, the increase in profit per cycle used is a better measure of the desirability of a partial order than the increase in profit per additional computer, so we divide the latter quantity by the number of cycles required to produce the type of computer requested by the RFQ and use the resulting values to choose which RFQ should be considered next. We consider cycles to be the limiting factor whenever the previous day's production used more than 90% of the available cycles.

The range of possible bid prices is discretized for the sake of efficiency. Even with fairly fine granularity, this bidding heuristic produces a set of bids in significantly less time than the 15 seconds allowed per simulated game day. The complete bidding heuristic is summarized in Algorithm 1.

4.4.3.4 Completing Production and Delivery

After applying the production scheduler to the current orders and RFQs, the Demand Manager is left with a 10-day production schedule, a record of how each order was filled, and a set of bids for the actual and predicted RFQs. The bids on actual RFQs can be sent directly to customers in their current form, and computers scheduled for delivery can be shipped. The Demand Manager then considers modifications to the production schedule to send to the factory for the next day. If there are no cycles remaining on the first day of the 10-day production schedule, the first day can be sent unchanged to the factory. Otherwise, the Delivery Manager shifts production from future days into the first day so as to utilize all cycles, if possible.

4.4.3.5 Production Beyond 10 Days

The components purchased by the Supply Manager depend on the component use projected by the Demand Manager. If we want to allow the possibility of ordering components more than 10 days in advance, the Demand Manager must be able to project its component use beyond the 10-day period for which it plans production. One possibility we considered was to extend this period and predict RFQs farther into the future. Another was to predict future computer and component prices by estimating our opponents' inventories and predicting their future behavior. Neither method provided accurate predictions of the future, and both resulted in large swings in projected component use from one day to the next. The Demand Manager thus uses a simple and conservative prediction of future component use.

The Demand Manager attempts to predict its component use for the period between 11 and 40 days in the future. Before 11 days, the components used in the 10-day production schedule are used as the prediction, and situations in which it is advantageous to order components more than 40 days in advance appear to be rare. The Demand Model is used to predict customer demand during this period, and the Demand Manager assumes that it will win, and thus need to produce, some fraction of this demand. This fraction ranges from zero during times of low demand to 1/6 during times of moderate or high demand, although the Demand Manager will not predict a higher level of component use than is possible given the available factory cycles.

4.5 The Supply Manager

The Supply Manager is responsible for purchasing components from suppliers based on the projection of future component use provided by the Demand Manager, and for informing the Demand Manager of expected component deliveries and replacement costs. In order to be effective, the Supply Manager must be able to predict future component availability and prices. The Supplier Model assists in these predictions.

4.5.1 Supplier Model

The Supplier Model keeps track of all information sent to and received from suppliers. This information is used to model the state of each supplier, allowing predictions to be made. The Supplier Model performs three main tasks: predicting component prices, tracking reputation, and generating probe RFQs to improve its models.

4.5.1.1 Price Prediction

To assist the Supply Manager in choosing which RFQs to send to suppliers, the Supplier Model predicts the price that a supplier will offer in response to an RFQ with a given quantity and due date. The Supplier Model requires an estimate of each supplier's existing commitments in order to make this prediction.

Algorithm 2 `Supplier Free Capacity Estimation`

Initialization: Create an array $freeCapacity$ of size 220 (the number of game days) to store estimates, with all entries set to 1 (all capacity free), and record the component's base price $base$.

Each day:

1. Input the current day d and a set of n offers sorted by due date. For $1 \leq x \leq n$, offer x is represented by due date due_x and price $price_x$.
2. Compute $free_x = 2(1 - price_x/base)$. This is the free capacity from today until due_x based on the pricing equation. Also, let $free_0 = 0$ and $due_0 = d$;
3. For $x = 1$ to n, compute $free'_x = \frac{free_x(due_x - d - 1) - free_{x-1}(due_{x-1} - d - 1)}{due_x - due_{x-1}}$. This is the fraction of free capacity between days due_{x-1} and $due_x - 1$.
4. For $x = 1$ to n, compute $estimate_x = \frac{\sum_{i=due_{x-1}}^{due_x - 1} freeCapacity[i]}{due_x - due_{x-1}}$. This is our estimate of that free capacity fraction.
5. Correct the daily estimates: For $x = 1$ to n:

   For $i = due_{x-1}$ to $due_x - 1$, $freeCapacity[i] \mathrel{+}= free'_x - estimate_x$

Recall that the price offered in response to an RFQ requesting delivery on a given day is determined entirely by the fraction of the supplier's capacity that is committed through that day. As a result, the Supplier Model can compute this fraction from the price offered. If two offers with different due dates are available, the fraction of the supplier's capacity that is committed in the period between the first and second date can be determined by subtracting the total capacity committed before the first date from that committed before the second. With enough offers, the Supplier Model can form a reasonable estimate of the fraction of capacity committed by a supplier on any single day.

For each supplier and supply line, the Supply Manager maintains an estimate of free capacity, and updates this estimate daily based on offers received. Using this estimate, the Supplier Model is able to make predictions on the price a supplier will offer for a particular RFQ. The update process for each supplier and supply line is summarized in Algorithm 2.

4.5.1.2 Reputation

When deciding which RFQs to send, the Supply Manager needs to be careful to maintain a good reputation with suppliers. Each supplier has a minimum acceptable purchase ratio, and the Supply Manager tries to keep this ratio above the minimum. The Supplier Model tracks the offers accepted from each supplier and informs the Supply Manager of the quantity of offered components that can be rejected from each supplier before the ratio falls below the minimum.

4.5.1.3 Price Probes

The Supply Manager will often not need to use the full five RFQs allowed each day per supplier line. In these cases, the remaining RFQs can be used as zero-quantity price probes to improve the Supplier Model's estimate of a supplier's committed capacity. For each supplier line, the Supplier Model records the last time each future day has been the due date for an offer received. Each day, the Supply Manager informs the Supplier Model of the number of RFQs available per supplier line to be used as probes. The Supplier Model chooses the due dates for these RFQs by finding dates that have been used as due dates least recently.

4.5.2 Supply Manager

The Supply Manager's goal is to obtain the components that the Demand Manager projects it will use at the lowest possible cost. This process is divided into two steps: first the Supply Manager decides *what* components will need to be delivered, and then it decides *how* best to ensure the delivery of these components. These two steps are described below, along with an alternative means of obtaining components.

4.5.2.1 Deciding What to Order

The Supply Manager seeks to keep the inventory of each component above a certain threshold. This threshold (determined experimentally) is 800, or 400 in the case of CPUs, and decreases linearly to zero between days 195 and 215. Each day the Supply Manager determines the deliveries that will be needed to maintain the threshold on each day in the future. Starting with the current component inventory, the Supply Manager moves through each future day, adding the deliveries from suppliers expected for that day, subtracting the amount projected to be used by the Demand Manager for that day, and making a note of any new deliveries needed to maintain the threshold. The result is a list of needed deliveries that we call *intended deliveries*. When informing the Demand Manager of the expected future component deliveries, the Supply Manager adds these intended deliveries to the actual deliveries expected from previously placed component orders. The idea is that although the Supply Manager has not yet placed the orders guaranteeing these deliveries, it intends to, and is willing to make a commitment to the Demand Manager to have these components available.

Because prices offered in response to short term RFQs can be very unpredictable, the Supply Manager never makes plans to send RFQs requesting delivery in less than five days. (One exception is discussed later.) As discussed previously, no component use is projected beyond 40 days in the future, meaning that the intended deliveries fall in the period between five and 40 days in the future.

4.5.2.2 Deciding How to Order

Once the Supply Manager has determined the intended deliveries, it must decide how to ensure their delivery at the lowest possible cost. We simplify this task by requiring that for each component and day, that day's intended delivery will be supplied by a single order with that day as the due date. Thus, the only decisions left for the Supply Manager are when to send the RFQ and which supplier to send it to. For each individual intended delivery, the Supply Manager predicts whether sending the RFQ immediately will result in a lower offered price than waiting for some future day, and sends the RFQ if this is the case.

In order to make this prediction correctly, the Supply Manager would need to know the prices that would be offered by a supplier on any future day. Although this information is clearly not available, the Supplier Model does have the ability to predict the prices that would be offered by a supplier for any RFQ sent on the current day. To enable the Supply Manager to extend these predictions into the future, for the 2005 agent we made the simplifying assumption that the price pattern predicted on the current day will remain the same on all future days. In other words, if an RFQ sent on the current day due in i days would result in a certain price, then sending an RFQ on any future day d due on day $d + i$ would result in the same price. This assumption is not entirely unrealistic due to the fact that agents tend to order components a certain number of days in advance, and this number generally changes slowly. Essentially, we are saying, "Given the current ordering pattern of other agents, prices are lowest when RFQs are sent x days in advance of the due date, so plan to send all RFQs x days in advance." (In Section 5.2 of the next chapter, I will discuss the addition of learning to predict changes in component prices.)

The resulting procedure followed by the Supply Manager is as follows. For each intended delivery, the Supplier Model is asked to predict the prices that would result from sending RFQs today with various due dates requesting the needed quantity. A price is predicted for each due date between 5 and 40 days in the future. If there are two suppliers, the lower price is used. If the intended delivery is needed in i days, and the price for ordering i days in advance is lower than that of any smaller number of days, the Supply Manager will send the RFQ. Any spare RFQs will be offered to the Supplier Model to use as probes.

The final step is to predict the replacement cost of each component. The Supply Manager assumes that any need for additional components that results from the decisions of the Demand Manager will be felt on the first day on which components are currently needed, i.e., the day with the first intended delivery. Therefore, for each component's replacement cost, the Supply Manager uses the lowest price found when considering the first intended delivery of that component, even if no RFQ was sent.

For each RFQ, a reserve price somewhat higher than the expected offer price is used. Because the Supply Manager believes that the RFQs it sends are the ones that will result in the lowest possible prices, all offers are accepted. If the reserve price cannot be met, the Supplier Model's predictions will be updated accordingly and the Supply Manager will try again the next day.

4.5.2.3 2-Day RFQs

As mentioned previously, the prices offered in response to RFQs requesting near-immediate delivery are very unpredictable. If the Supply Manager were to wait until the last minute to send RFQs in hopes of low prices, it might frequently end up paying more than expected or be unable to buy the components at all. To allow for the possibility of getting low priced short-term orders without risk, the Supply Manager sends RFQs due in 2 days, the minimum possible, for small quantities in addition to what is required by the intended deliveries. If the prices offered are lower than those expected from the normal RFQs, the offers will be accepted.

The size of each 2-day RFQ depends on the need for components, the reputation with the supplier, and the success of past 2-day RFQs. Because the Supply Manager may reject many of the offers resulting from 2-day RFQs, it is possible for the agent's reputation with a supplier to fall below the acceptable purchase ratio. The Supplier Model determines the maximum amount from each supplier that can be rejected before this happens, and the quantity requested is kept below this amount.

The Supply Manager decides whether to accept an offer resulting from a 2-day RFQ by comparing the price to the replacement cost and the prices in offers resulting from normal RFQs for that component. If the offer's price is lower than any of these other prices, the offer is accepted. If the quantity in another, more expensive offer is smaller than the quantity of the 2-day RFQ, then that offer may safely be rejected.

The 2-day RFQs enable the agent to be opportunistic in taking advantage of short-term bargains on components without being dependent on the availability of such bargains.

4.6 2005 Competition Results

Out of 32 teams that initially entered the 2005 TAC SCM competition, 24 advanced past a seeding round to participate in the finals, held over three days at IJCAI 2005. On each day of the finals, half of the teams were eliminated, until six remained for the final day. Game outcomes depended heavily on the six agents competing in each game, as illustrated by the progression of scores over the course of the competition. In the seeding round, TacTex won with an average score of \$14.9 million, and several agents had scores above \$10 million. Making a profit was much more difficult on the final day of competition, however, and TacTex won with an average score of only \$4.7 million. The second highest average score was \$1.6 million, and three agents (each of which averaged at least \$6 million in the seeding round) lost money. Scores of the final round are shown in Table 4.2.

Due to the complexity of the TAC SCM scenario and the vast number of decisions that must be made during a single game, it is difficult to isolate the factors that contributed to TacTex's success by analyzing game results. When comparing

Rank	*Agent*	*Average Profit*
1	TacTex-05	$4.74M
2	Southampton-SCM	$1.60M
3	Mertacor	$0.55M
4	DeepMaize	*-$0.22M*
5	MinneTAC	*-$0.33M*
6	Maxon	*-$1.99M*

Table 4.2: Results of the 2005 final round

purchases of individual component types or sales of individual computer types on a day-by-day basis[1], it does not appear that TacTex obtained significantly lower purchase prices or significantly higher sales prices than other competitive agents. This fact suggests the possibility that TacTex was better able to focus on the types of computer that were most profitable at any point in time given the component and computer prices then present in the market. Two observations potentially related to this hypothesis are that TacTex tends to carry smaller component inventories throughout the game than many of its competitors, and also appears more flexible in its choice of when to buy components, showing a willingness to purchase components only a short time in advance of their use. An agent that buys components at the last possible moment may be better able to match its purchases to current customer demand. There is the risk, however, that the agent might wait too long and be unable to purchase components at a reasonable price, or at all. This trade-off, and other possible factors related to TacTex's success, are explored in the next section.

4.7 Experiments

I now present the results of controlled experiments designed to measure the impact of individual components of TacTex on its overall performance. In each

[1] using the CMieux toolkit [8]

experiment, two versions of TacTex compete: one unaltered agent (which I will call the base agent) that matches the description provided previously, and one agent that has been modified in a specific way. Each experiment includes 30 games. The other four agents competing–Mertacor-05, MinneTAC-05, GoBlueOval, and RationalSCM–are taken from the TAC Agent Repository[2], a collection of agents provided by the teams involved in the competition.

Experimental results are shown in Table 4.3. Each experiment is labeled with a number. The columns represent the averages over the 30 games of the total score (profit) along with standard deviations, percent of factory utilization over the game (which is closely correlated with the number of computers sold), revenue from selling computers to customers, component costs, storage costs, penalties for late deliveries, and the percent of the games in which the altered agent outscored the base agent. The final column indicates whether the difference in score observed between the two agents is statistically significant with 99% confidence according to a paired t-test. The first row, experiment 0, is provided to give perspective to the results of other experiments. In experiment 0, two base agents are used, and all numbers represent the actual results obtained. In all other rows, the numbers represent the *differences* between the results of the altered agent and the base agent (from that experiment, not from experiment 0). In general, the results of the base agents are close to those in experiment 0, but there is some variation due to differences between games (e.g. customer demand), and due to the effects of the altered agent on the economy.

I first present experiments designed to measure the importance of prediction accuracy in our predictive planning approach. Then I examine the sensitivity of our agent to some of its parameters, particularly those related to the important decision of when to purchase components.

[2] `http://www.sics.se/tac/showagents.php`

Exp. #	*Score*	*Util.*	*Revenue*	*Costs*	*Storage*	*Penalties*	*Win %*	*Sig.?*
0	\$2.54M	89%	\$111.25M	\$106.14M	\$1.91M	\$.36M	**-**	**-**
1	-2.46 ± 2.65	-1%	-2.25	-5.4	+.70	-.20	0%	**Y**
2	-.19 ± .68	0%	-.08	+.17	+.01	-.08	30%	**N**
3	-.75 ± 1.23	-3%	-4.57	-3.56	+.07	-.30	20%	**Y**
4	+.05 ± 1.28	+1%	+2.45	+2.36	+.01	+.02	40%	**N**
5	+.65 ± .99	+2%	+2.09	+1.58	-.14	+.03	87%	**Y**
6	-2.36 ± 2.75	-5%	-7.01	-4.18	-.54	+.06	7%	**Y**
7	-3.88 ± 4.74	-44%	-52.24	-47.97	-.81	+.50	7%	**Y**
8	+.10 ± 1.65	-28%	-31.80	-31.41	-.69	+.33	53%	**N**
9	+1.29 ± 1.73	-10%	-12.23	-13.07	-.49	+.17	87%	**Y**
10	-1.66 ± 1.99	+4%	+4.65	+5.84	+.45	-.11	10%	**Y**
11	-2.70 ± 3.02	+4%	+5.89	+7.62	+.86	-.18	3%	**Y**
12	-4.34 ± 4.53	+6%	+6.40	+10.12	+.58	-.20	0%	**Y**
13	-2.13 ± 3.23	-31%	-40.01	-38.55	-.24	+.95	13%	**Y**
14	-.46 ± 3.83	-18%	-25.94	-25.88	-.08	+.51	50%	**N**
15	-.18 ± .80	0%	-.54	-.43	-.03	+.12	40%	**N**
16	-.20 ± .83	+1%	+1.51	+1.70	+.04	-.02	37%	**N**
17	-1.25 ± 1.74	+4%	+5.47	+6.63	+.22	-.19	10%	**Y**
18	+.03 ± 3.92	-15%	-22.42	-22.65	-.04	+.23	70%	**N**
19	+.78 ± 1.76	-9%	-12.26	-13.15	+.03	+.12	73%	**Y**
20	+.46 ± .88	-2%	-3.14	-3.58	-.04	+.01	73%	**Y**
21	-.41 ± .89	+1%	+1.18	+1.51	+.04	+.02	40%	**Y**
22	+.11 ± 2.73	-19%	-24.85	-24.64	-.45	+.26	66%	**N**
23	-6.76 ± 8.00	-48%	-66.96	-60.52	-.38	+.57	0%	**Y**
24	-2.05 ± 2.29	-4%	-5.62	-3.54	-.12	+.06	0%	**Y**

Table 4.3: Experimental results for one altered and one unaltered version of TacTex in 30 games. Columns include the total score (with standard deviations), how often the altered agent outscored the unaltered agent, and an indicator of statistical significance. Numbers represent millions of dollars. In experiment 0, provided to provide perspective, no alteration is made to TacTex, and numbers represent the actual results. In all other experiments, numbers represent the difference between the altered and unaltered agent.

4.7.1 The Three Predictor Modules

The first three sets of experiments probe the sensitivity of the agent to changes in the predictor modules.

4.7.1.1 Offer Acceptance Predictor

Recall that for each customer RFQ a function is generated mapping a price offered to the probability of acceptance. This function is generated by multiplying the result of linear regression by a day factor. In experiment 1, no day factors are used, and the score decreases considerably. In experiment 2, day factors are used, but instead of using linear regression to find probabilities across prices, a single price is chosen at which the RFQ is expected to be won. The price chosen is the greater of 95% of the previous day's high price for the computer type, and the previous day's low price. This quantity corresponds roughly to the average selling price of a computer. For any offer below this price, the prediction is made that the offer will be accepted with probability 1, before the day factor is applied. The results show a small, non-significant difference between the use of this heuristic and the use of linear regression.

It appears that the use of day factors is the key to the success of the Offer Acceptance Predictor. The day factors serve two roles, however: measuring the impact of due date on offer acceptance, and serving as a feedback mechanism to ensure that the number of orders received is in line with the predictions. To measure the relative importance of these two roles, I replace the day factors in experiment 3 with a single multiplier to be used regardless of an RFQ's due date. Like an all-inclusive day factor, the multiplier represents the ratio between all actual and expected orders (i.e. those predicted from the previous days). Linear regression is used as before. The results show the single multiplier to be less effective than the day factors, but much more effective than nothing at all, as in experiment 1. Thus, while considering due dates is of some value in predicting offer acceptance, it appears the feedback role is the more important aspect of the day factors.

4.7.1.2 Demand Model

In experiment 4, I investigate the value of the Demand Model by ignoring its predictions and instead assuming that the number of RFQs received on any day in the future will be the same as the number received on the current day. Surprisingly, this has an insignificant effect on performance, and we have been unable to determine why. It may be the case that demand changes slowly enough that accurate predictions are not necessary.

4.7.1.3 Supplier Model

The predictions of prices that will be offered in response to RFQs cannot simply be "turned off" as the predictions were in the previous experiments, because the behavior of the Supply Manager is dependent upon comparisons of prices. Instead, in experiment 5 I measure the impact of accuracy in the predictions of component offer prices by the Supplier Model by adding noise to the predictions. Each predicted offer price for a particular RFQ under consideration is multiplied by a number chosen randomly between 0.9 and 1.1. Amazingly, this actually *improves* performance.

One explanation for this result is that component costs tend to decrease over the course of a game. Recall that the Supply Manager will only order components due on day d if there is no n for which waiting to order on day $d - n$ is predicted to result in a lower price. Because the Supplier Model assumes that price patterns will remain the same in the future, it is likely to overestimate the cost of waiting to purchase components. The presence of noise makes it less likely that a long-term request would be predicted to result in the lowest offer price–for some value of n, the added noise will possibly make waiting until day $d - n$ appear more attractive. This explanation is supported by the results of Section 5.2, in which I explore the addition of learning to predict changes in future component prices.

Another explanation is that by waiting longer to order components, TacTex may have a better idea of what components it actually needs. This explanation is supported by the results of the next section.

4.7.2 Experiments with the Supply Manager

The results to this point highlight the importance of the Offer Acceptance Predictor within TacTex and demonstrate that the Demand Predictor is, at least in the economy under consideration, not important. The results of experiment 5, along with the hypothesis in the discussion of the competition that TacTex's success may be related to its choice of when to purchase components, suggest further experiments into the strategy used by the Supply Manager. I now present experiments that examine TacTex's sensitivity to several parameters in the Supply Manager.

4.7.2.1 2-day RFQs

In experiment 6, 2-day RFQs are not used, preventing the agent from taking advantage of bargains on components requested in the short term. The resulting decrease in score indicates that 2-day RFQs are an important factor in the agent's ability to acquire components at low prices. In addition, the decrease in factory utilization suggests that components purchased from 2-day RFQs do not simply take the place of components that would otherwise have been purchased further in advance. Apparently the prices obtained are sufficiently low to make additional production profitable.

4.7.2.2 Inventory Threshold

The next set of experiments examines the impact of the inventory threshold used. Normally, the Supply Manager attempts to maintain an inventory of at least

800 components of each type beyond the projected use. In experiments 7, 8, 9, 10, and 11, inventory thresholds of 100, 200, 400, 1200, and 1600 are used, respectively. An agent able to perfectly predict future component needs and availability would have no need to maintain surplus inventory, and could plan so that component deliveries would arrive just in time to be used. Maintaining a large inventory can therefore be seen as a way of dealing with inaccuracies in predictions, preventing the lost revenue and penalties that can come from component shortages at the cost of higher storage costs and possibly unnecessary component purchases. This tradeoff is clearly seen in the results. As the inventory threshold increases, factory utilization and revenue increases and penalties decrease, but storage costs increase, and the additional production is not necessarily profitable.

This last fact is somewhat surprising, because it is not immediately clear why simply holding a larger inventory would cause an agent to sell computers at a loss. One possible explanation is that component costs tend to decrease over the course of a game, as mentioned above, and therefore an agent that builds up a large surplus inventory early in the game is buying these components at their most expensive, but this cannot account for the entire difference. An analysis of the game logs shows that for the agents using a threshold below the standard 800, the reduced factory utilization is usually caused by a shortage of components, and not a voluntary decision to not produce. Component shortages can only occur when the agent is unable to obtain components that it had planned to buy, indicating that predictions of component availability were too high. In this situation, predicted replacement costs would be too low, possibly causing the agent to sell computers when it is not actually profitable to do so. This idea is supported by the observation that during times when the base agent is producing computers and the altered agent is not, the base agent's score is often decreasing. The results of this experiment thus suggest that improvements to the component price predictions may be needed in certain situations.

Of the inventory thresholds tried in this experiment (as well as the default 800), 400 clearly resulted in the highest profits. In succeeding years' agents, this parameter was tuned more carefully, and a threshold of around 400 was generally found to be optimal.

4.7.2.3 Reduced RFQ Flexibility

As mentioned previously, flexibility in the choice of how far in advance to buy components appears to be a distinguishing characteristic of TacTex. In experiments 12, 13, and 14, I remove this flexibility and consider three simple alternative methods of deciding when components should be ordered. In experiment 12, no attempt is made to wait for the best day to send RFQs, and RFQs are sent for all needed components immediately. In experiment 13, components are always requested ten days in advance of anticipated need. In experiment 14, components are requested five days in advance. None of these strategies appear effective. In experiment 12, components are purchased in higher quantities and at higher prices than usual, resulting in a very poor score. In experiments 13 and 14, the game logs show that the prices paid for components are not much higher than the prices paid by the base agent. Good prices can apparently be found on these dates much of the time, but not always, as indicated by the decreased factory utilization. Somewhat surprisingly, component availability appears higher when components are requested five days in advance rather than ten, and the relatively small decrease in score in experiment 14 despite the large decrease in factory utilization suggests that the situations in which production is reduced may be those in which it is least profitable, similar to the effect observed with the previous set of experiments.

4.7.2.4 Range of Days Considered

As the choice of when to request components appears important, the next two sets of experiments are designed to measure the effect of restricting the number of days in advance in which the Supply Manager can plan to request components, although to a lesser degree than the previous experiments. Recall that the Supply Manager will normally request components at least five days in advance of anticipated need, and at most 40 days in advance. In experiments 15, 16, and 17, the minimum number of days is changed to four, seven, and ten, respectively, and in experiments 18, 19, 20, and 21, the maximum number of days is changed to 10, 20, 30, and 50, respectively. From experiments 15, 16, and 17, it appears that the best prices can often be found by waiting until fewer than ten days remain. The results of experiments 18 through 21 are somewhat more surprising. It appears that ordering too far in advance, in particular beyond 20 days, is detrimental to performance. This could be due either to incorrectly predicting long term prices (as suggested by experiment 5), or buying unneeded components due to incorrect projected future component use. Data from the game logs suggests the latter. Prices paid do not seem to differ significantly between the base agent and altered agents. It appears that similarly good prices can be found when ordering either a long or short distance in advance, and that the agents that are restricted from ordering far in advance are able to make purchases that better reflect the current customer demand. Unlike the experiments involving inventory thresholds, it appears that the reduction in factory utilization is planned (not due to component shortages) and occurs during periods of low customer demand, while the base agent is busy using components it purchased in advance before demand decreased.

The success of the changes made in experiments 9 and 19 prompt experiment 22, in which an inventory threshold of 400 and a maximum request distance of 20 days are used. From the result, it appears that these two changes are not complementary, and that an agent that is unable to request components far in advance may need a larger inventory threshold to offset the risk of being unable to regularly obtain components.

4.7.2.5 The Effect of Opponent Strategies

Based on the previous experiments, it is tempting to conclude that any change that results in TacTex buying components a shorter distance in advance of their use will be beneficial. Even the simple agent in experiment 15 that is required to request all components exactly five days in advance performs nearly as well as the base agent. It is important to consider that the attractiveness of short term purchasing may simply be a feature of the set of agents competing. If all six agents attempted to use such a short term strategy, would it remain effective? To test this, we replaced the four additional agents previously used with four copies of TacTex modified to request components at most ten days in advance. In experiment 23, the agent restricted to requesting all components five days in advance is tested in this environment, and in experiment 24, the agent limited to requesting components at most 20 days in advance is tested. In both cases, the results are much worse than the results against the previously used group of agents. Thus, there must be some value in maintaining the option of long-term component requests. Based on these experiments and experiment 22, in succeeding years' agents we retained the 40 day ordering window.

4.8 Summary

This chapter presented the TacTex agent for TAC SCM. TacTex operates by making predictions about various quantities (prices, customer demand, component deliveries, etc.) and then optimizing with respect to those predictions. This chapter focused on the optimization aspects, which have remained mostly the same since 2005. The predictive methods described in this chapter were largely heuristic in nature. Competition and experimental results show that these methods are sufficient for strong agent performance, but there is nevertheless significant room for improvement, as will be shown in the following chapter.

Chapter 5

Learning for Price Prediction in TAC SCM

Chapter 4 described the TacTex agent for TAC SCM. As shown in the experiments of that chapter, the price predictions made by the Offer Acceptance Predictor and Supplier Model are important to the performance of TacTex, but there is room for improvement, particularly in the area of predicting how prices will change in the future. In this chapter I describe improvements to these two agent components based on using machine learning to model price changes. I also present a comparison of TacTex's prediction accuracy to the accuracy of other agents and explore the importance of the choice of training data. This chapter fulfills Contribution 2 for TAC SCM, as outlined in Section 1.2.

5.1 Learning for Computer Price Prediction

This section describes improvements to computer price prediction, which were the main improvements in the 2006 agent.

5.1.1 Particle Filter for Price Distribution Estimation

Recall that the Offer Acceptance Predictor from the 2005 agent described in Chapter 4 uses a heuristic to obtain a linear function representing the probability of an offer on an RFQ being accepted at a given price. This function is in fact the cumulative distribution function corresponding to a probability density function that is uniform over some range and represents the probability of a given price being the lowest bid of any competing manufacturer (not including TacTex). An inspection of each day's low bids for each type of computer in a typical completed game suggests that these prices instead tend to follow a normal distribution. We developed an improved version of the Offer Acceptance Predictor that estimates these distributions using all available information on computer prices.

To estimate these distributions during a game, the Offer Acceptance Predictor makes use of a separate particle filter (specifically a Sampling Importance Resampling filter [1]) for each computer type. A particle filter is a sequential Monte Carlo method that tracks the changing state of a system by using a set of weighted samples (called particles) to estimate a posterior density function over the possible states. The weight of each particle represents its relative probability, and particles and weights are revised each time an observation (conditioned on the current state) is received (see Section 7.1.2 for more information on particle filtering). In this case, each of the 100 particles used per filter represents a normal distribution (indicating the probability that a given price will be the winning price on the computer) with a particular mean and variance. At the beginning of each game, weights are set equally and each particle is assigned a mean and variance drawn randomly from a distribution that is generated by analyzing the first day prices from a large data set of past games. (The source of this data set will be described below.) Each succeeding day, a new set of particles is generated from the old. For each new particle to be generated, an old particle is selected at random based on weight, and the new particle's estimate of mean and variance are set to those of the old particle plus small changes, drawn randomly from the distribution of day-to-day changes seen in the data set of past games. The new particles are then reweighted, with the weight of each particle set to the probability of the previous day's price-related observations occurring according to the distribution represented. These observations consist of the reported highest and lowest winning prices and the acceptance or rejection of each offer made to a customer by TacTex for the given type of computer. Finally, the weights are normalized to sum to one. The distribution of winning prices predicted by the particle filter is simply the weighted sum of the individual particles' distributions, and from this distribution the function mapping each possible bid price to a probability of acceptance can be determined.

5.1.2 Future Price Changes

In order to maximize revenue from the computers sold, the Demand Manager needs to consider not only the prices it will offer in response to the current day's RFQs, but also what computers it will wish to sell on future days. As described in Section 4.4, the Demand Manager plans ahead for 10 days and considers future as well as current RFQs when making offers. It is therefore important for the Offer Acceptance Predictor to be able to predict future changes in computer prices. To illustrate why this is important, Figure 5.1 shows the prices at which one type of computer sold during a single game of the 2006 finals. For each day, points representing one standard deviation above and below the average price are plotted. On most days, there is clearly little variance among the winning prices, but prices often change drastically over the course of a few days. This fact suggests that it may be even more valuable to be able to predict future changes in price than to predict the distribution of winning prices on a single day. By simply selling a computer a few days earlier or later, it might be possible for the Demand Manager to significantly increase the price it obtains.

To make these predictions of price changes, the Offer Acceptance Predictor performs machine learning on data from past games. Each training instance consists of 31 features representing data available to the agent during the game, such as the date, estimated levels of customer demand and demand trend, and current and recent computer prices. The full list of features is shown in Table 5.1. All prices are expressed as a fraction of the base price. One instance is generated for each computer type and game day. The label for each instance is the amount by which the average price changes in ten days. Once the Offer Acceptance Predic-

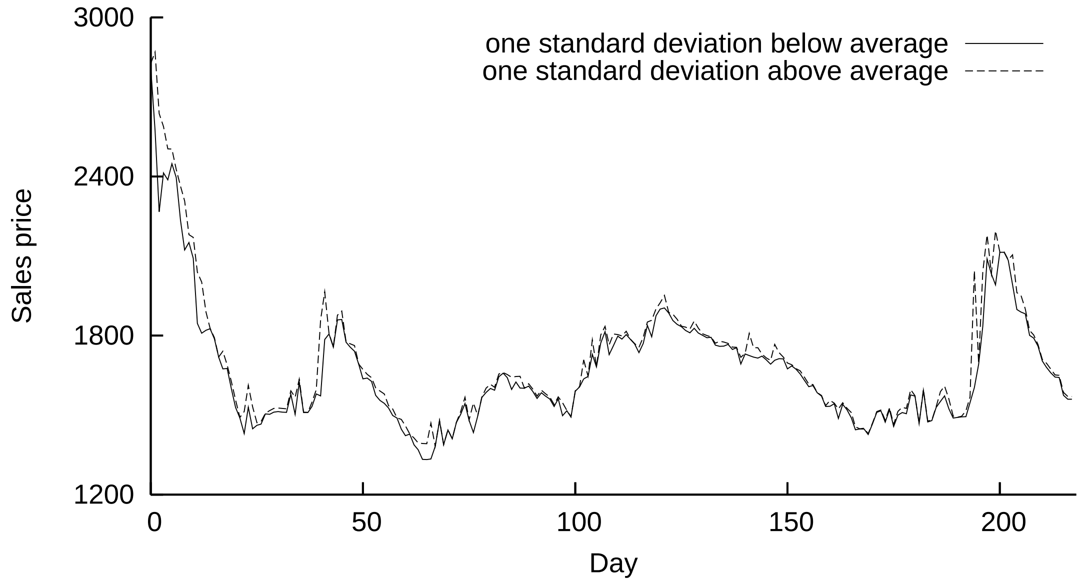

Figure 5.1: Average prices at which one type of computer sold during one game of the 2006 finals. One standard deviation above and below the average is shown.

tor has learned to predict this quantity, it can predict the change in average price for any day between zero and ten days in the future through linear interpolation. (This approach appears to work as well as trying to learn price changes for each day individually, and reduces learning time.) No effort is made to predict changes in the shape of the distribution, i.e., the variance. Thus, to generate an offer acceptance function for a future RFQ, the Offer Acceptance Predictor simply shifts the predicted distribution over winning prices up or down depending on the predicted change in average price, and bases the acceptance function on this modified distribution.

In order to train the price change prediction model, a learning algorithm and source of training data must be chosen. After experimenting with various algorithms from the WEKA machine learning package [110], we selected additive regression with decision stumps, an iterative method in which a decision stump is

- current date
- demand (in computer's segment)
- trend (in computer's segment)
- predicted demand in 10 days
- demand (10, 20) days ago
- predicted change in demand in 10 days
- change in demand from (10, 20) days ago
- total demand (in all segments)
- total trend (in all segments)
- predicted total demand in 10 days
- total demand (10, 20) days ago
- predicted change in total demand in 10 days
- change in total demand from (10, 20) days ago
- predicted price today
- price 10 days ago
- change in price over (1, 5, 10) days
- predicted average price (all computers)
- average price 10 days ago
- change in average price over (1, 5, 10) days
- predicted price in 10 days using linear regression over last (5, 10) days
- daily price change using linear regression over last (5, 10) days

Table 5.1: Features for learning changes in computer prices.

repeatedly fit to the residual from the previous step[1]. For training data, we could have used data from games in the competition, but instead we ran a large number of games of our own using both variations of TacTex-06 and other agents taken from the TAC Agent Repository. This decision was based on the fact that there is limited data available for training during the competition, while generalization

[1]M5P trees gave nearly identical performance, but we chose to use additive regression as the models it generated were significantly smaller. For the work described in later chapters, we switched to M5P trees for this learning problem due to their better performance when paired with other algorithms such as boosting.

Rank	*Agent*	*Average Profit*
1	TacTex-06	\$5.85M
2	PhantAgent	\$4.15M
3	DeepMaize	\$3.58M
4	Maxon	\$1.75M
5	Botticelli	\$0.48M
6	MinneTAC	*-\$2.70M*

Table 5.2: Results of the 2006 final round

between models trained on different groups of agents appeared to be reasonably good in our testing. I will revisit this decision in Section 5.4.

5.1.3 2006 Competition Results

Out of 21 teams that participated in the final round of the 2006 TAC SCM competition, held over three days at AAMAS 2006, six advanced to the final day of competition. After 16 games between these agents, TacTex had the highest average score, followed closely by PhantAgent and DeepMaize. Scores are shown in Table 5.2. Both PhantAgent and DeepMaize were much improved over their 2005 counterparts, and would very likely have beaten the previous year's champion, TacTex-05, if it had competed unchanged. It thus appears that the improvements present in TacTex-06 were an important part of its victory. Although it is difficult to assign credit for an agent's performance in the competition to particular components, we can make some observations that support this hypothesis.

Figure 5.2 shows the average, over all 16 games on the final day of the competition, of the profit earned per game day for the top three agents. Daily profit is computed by determining what computers were delivered to customers each day and which components in inventory went into those computers, and then subtracting costs from revenue. TacTex-06 clearly had the highest daily profits over the first 70 days of the game, and after this point profits were roughly equal for all three agents. The difference in profits appears to be accounted for by higher

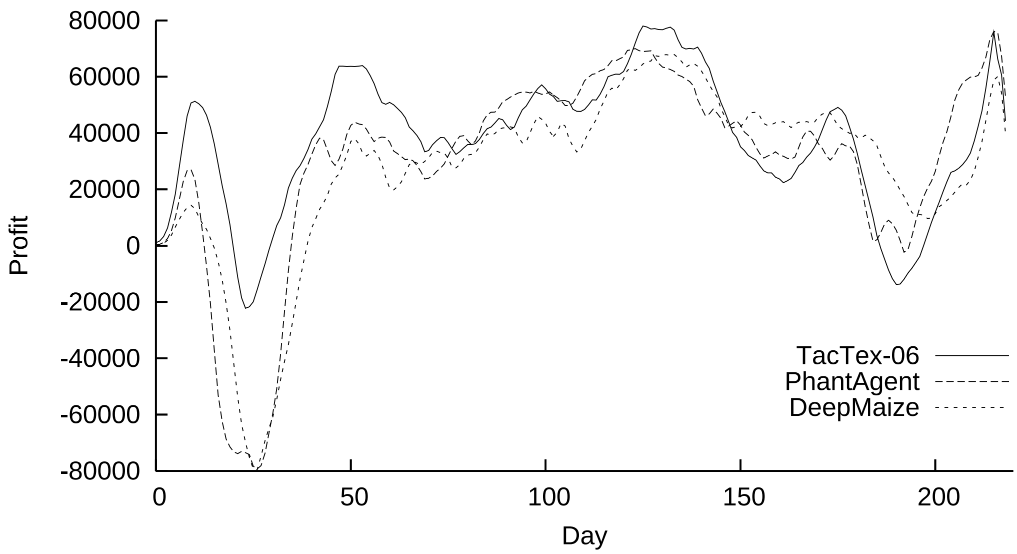

Figure 5.2: Daily profits for the top three agents on the final day of the 2006 competition, averaged over all 16 games.

revenue per computer. During the first 70 days of each game, TacTex-06 sold about as many computers as PhantAgent and DeepMaize while paying roughly the same costs for components, but TacTex-06 almost always had a much higher average sales price for each type of computer. After day 70, TacTex-06 still had somewhat higher average computer prices, but these were offset by higher component costs than the other two agents paid.

The ability of TacTex-06 to sell computers at higher prices appears to be due to its attempt to predict future changes in computer prices and react accordingly. During the competition, TacTex-06 could often be seen building up its inventory of completed computers before prices rose or selling off its inventory as prices peaked, while such behavior among other agents was less visible. This behavior can explain not only the fact that TacTex-06 sold computers at higher prices, but also the fact that the advantage was especially large in the first portion of each game. To see why, consider Figure 5.1. For this particular game and computer type, prices began very high, then fell rapidly before recovering somewhat. This pattern is actually very common. Agents begin with no components or computers in inventory, and the supply of computers is thus much smaller than the demand in the beginning of each game. As agents obtain components and begin selling computers, prices usually drop rapidly. Due to the rapid changes in computer prices and the predictability of this pattern, the attempts by TacTex-06 to predict and exploit changes in prices are particularly effective in this period of the game.

5.1.4 Computer Price Prediction Experiments

I now present the results of controlled experiments designed to measure the impact of individual components of TacTex on its overall performance. As before, in each experiment two versions of TacTex compete: one unaltered agent that matches the description provided previously, and one agent that has been modified in a specific way. Each experiment involves 30 games. The other four agents competing–Mertacor, DeepMaize, MinneTAC, and PhantAgent (all versions from 2005)–are taken from the TAC Agent Repository. (Experiments against different combinations of agents appear to produce qualitatively similar results.)

Experimental results are shown in Table 5.3. Each experiment is labeled with a number. The columns represent the averages over the 30 games of the total score (profit), percent of factory utilization over the game, revenue from selling computers to customers, component costs, and the percentage of games in which the altered agent outscored the unaltered agent. In every experiment, the difference between the altered and unaltered agent is statistically significant with 99% confidence according to a paired t-test.

As before, the first row, experiment 0, is provided to give perspective to the results of other experiments. In experiment 0, two unaltered agents are used, and all numbers represent the actual results obtained. In all other rows, the numbers represent the *differences* between the results of the altered agent and the unaltered

Exp. #	*Description*	*Score*	*Util.*	*Revenue*	*Costs*	*Win %*
0	no changes	$7.28M	83%	$104.7M	$94.5M	-
1	no computer price change prediction	-3.51	-1%	-4.50M	-.70M	0%
2	no particle filter	-1.97	-7%	-10.05M	-8.03M	0%
3	no particle filter or prediction	-3.93	-6%	-10.99M	-6.83M	0%
4	heuristic price change prediction	-1.74	0%	-1.14M	-.64M	13%

Table 5.3: Experimental results for one altered and one unaltered version of TacTex. Columns represent the total score, percent of factory utilization, revenue from customers, component costs, and how often the altered agent outscored the unaltered agent. Numbers represent millions of dollars. In experiment 0, provided to place other experiments' results in perspective, no alteration is made to TacTex, and numbers represent the actual results. In all other experiments, numbers represent the difference between the altered and unaltered agent. In each experiment, the difference between the altered and unaltered agent is statistically significant.

agent (from that experiment, not from experiment 0). Again, the results of the unaltered agents are close to those in experiment 0, although there is some variation.

In experiment 1, the altered agent always predicts that future computer prices will remain unchanged. Not surprisingly, the result is a large decrease in revenue and score. The decrease in score is almost twice as large as the margin of victory for TacTex-06 in the 2006 competition ($1.8 million), adding more weight to the claim that the prediction of future price changes played a large role in the winning performance.

In experiment 2, the particle filter used to generate predictions of offer acceptance is replaced with the linear heuristic that was used in TacTex-05. The experiment shows that the particle filter approach is an improvement over this heuristic. The large drop in factory utilization in the altered agent is surprising. Experiment 3 shows the result when the changes of experiments 1 and 2 are combined: the agent makes no predictions of future price changes and uses the linear heuristic instead of the particle filter. The score is only slightly worse than in experiment 1, suggesting that the benefits of using the particle filter are more pronounced when price changes are predicted. It is possible that the more detailed and precise predictions of offer acceptance generated from the particle filter are necessary for the agent to effectively make use of the predictions of future price changes.

In experiment 4, the learned model of price changes is replaced with a heuristic that performs linear regression on the average computer price over the last ten days, and extrapolates the trend seen into the future to predict price changes. Although the heuristic's predictions are reasonably accurate, the performance of the altered agent is about midway between that of the unaltered agent and that of the agent from experiment 1 that makes no predictions at all, demonstrating the value of learning an accurate model.

5.2 Learning for Component Price Prediction

Recall that the Supply Manager needs to predict the price of placing an order m days from now due in $m + n$ days for various values of m and n. The Supplier Model described in Section 4.5.1 makes this prediction by estimating the prices for placing an order on the current day due in n days and assuming that this price does not change (thus ignoring m). As with computer prices, we can use machine learning on data from past games to predict changes in component prices. In this case, we perform learning using 29 features representing data available to the agent during the game, including information about dates and past prices. The full list of features is shown in Table 5.4. All prices are expressed as a fraction of the base price. One instance is generated for each RFQ sent to a supplier, excluding 2-day RFQs. The label for each instance is the difference between the original price prediction and the true offer price. Again, we experimented with the regression algorithms available in WEKA, and in this case M5P model trees were the best performing algorithm.

- current date
- due date
- date RFQ will be sent - date
- due date - date RFQ will be sent
- predicted price
- change in predicted price over last 20 days
- how many days in advance lowest price expected
- if lowest price expected for ordering now
- total demand
- total demand trend
- change in total demand over last 20 days
- predicted change in total demand over next 20 days
- sum of demand for computers using component
- sum of demand trends for computers using component
- fraction of all demand using this component
- if component is a CPU
- last reported supplier capacity
- predicted prices for orders due in (5, 10, 15, 20, 30, 40) days
- change over last 20 days in predicted prices for orders due in (5, 10, 15, 20, 30, 40) days

Table 5.4: Features for learning changes in component prices.

Exp. #	*Description*	*Score*	*Util.*	*Revenue*	*Costs*	*Win %*
1	no component price change prediction	-1.07	+0%	+.05M	+.96M	3%

Table 5.5: Experimental results for the component price change predictor.

We trained models for predicting component price changes on the same games used to train the computer price model, and performed an experiment against the same set of agents used in the experiments of Section 5.1.4. The results are shown in Table 5.5, with the same columns as the experiments of the previous section. In this case both agents included the improvements to the Offer Acceptance Predictor of the previous section, but one agent did not predict changes in component prices. This agent had nearly identical revenue and utilization, but paid almost $1 million more for components. The difference was statistically significant with 99% confidence according to a paired t-test. Thus, predicting changes in component prices can significantly reduce the prices an agent pays for components.

This experiment also supports the hypothesis of Section 4.7 that inaccurate component price predictions were in large part responsible for the problems with long term procurement in the experiments of that section. Indeed, after adding the predictor, both randomly perturbing price predictions and reducing the ordering window from the 40 day limit harm agent performance.

As the experiments on TacTex thus far have demonstrated, predicting prices accurately is an important part of successful supply chain management. Not surprisingly, many agents designed for TAC SCM depend on similar price predictions. In the next section, I discuss a side-competition to the main SCM competition that we designed to compare the accuracy of the price predictions of different agents.

5.3 The SCM Prediction Challenge

To be competitive in TAC SCM, an agent must be able to successfully perform a number of interrelated tasks. While this fact contributes to the complexity and realism of the scenario, it can also make it difficult to determine the relative effectiveness of agent components in isolation. To address this issue, in 2007 two challenges were introduced and run in addition to the full SCM game, each designed to measure an agent's performance on one specific task: a Procurement Challenge, and a Prediction Challenge. This section focuses on the Prediction Challenge, which we designed. I first give the specification of this challenge. Then I present the 2007 results, along with an analysis of how the predictions of the challenge participants compare to each other. Finally, I briefly discuss the 2008 results.[2]

5.3.1 Challenge Specification

While the methods used by different SCM agents to manage the supply chain vary considerably, many of these agents share a similar design at a high level - they divide the full problem into a number of smaller tasks and then solve these tasks using decision theoretic approaches based on maximizing utility given various predictions about the economy. The success of an agent thus depends on both the accuracy of the many kinds of predictions it makes and the manner in which these predictions are used, making it difficult to assign credit to individual agent components. To give a concrete example, suppose that based on available statistics from past games and the current one, agent A predicts that it will be able to sell one type of computer for \$2000 on day 45, and agent B predicts that it will be able to sell that computer for \$1900. They then make component purchases, plan manufacturing, and commit to customer orders based on these and other predictions. Ultimately, agent A wins. Is it safe to draw conclusions about the accuracy of these predictions based on this outcome? No.

The goal of the Prediction Challenge is to allow a head to head comparison of agents' prediction accuracy without concern for how these predictions are used. In the example above, if we had recorded the predictions and then observed on day 45 that the specific type of computer sold for an average price of \$1870, we could say that agent B made a more accurate prediction. This is exactly what takes place in the Prediction Challenge. There are many quantities for which agents may make predictions, such as customer demand, the probability that a particular offer to a customer will be accepted, and supplier capacities. However, the Prediction Challenge focuses only on those predictions that can be expressed in the form of

[2] Software, results, and the complete specifications are available from the Prediction Challenge website: `http://www.cs.utexas.edu/~TacTex/PredictionChallenge`

a price, namely component prices and computer prices. As agents need to be able to make predictions about future prices as well as current prices in order to plan effectively, the accuracy of predictions for both current and future prices is measured. There are thus four prediction categories in the Prediction Challenge: current and future computer prices, and current and future component prices.

Instead of making predictions about live TAC SCM games in which they are participating, participants in the challenge make predictions on behalf of another agent called the SCMPredictionAgent (or PAgent for short). (For clarity, I will refer to the manufacturing agents that participate in SCM games as *agents*, and the prediction agents participating in the Prediction Challenge as *participants*.) Before the competition, the organizers of the challenge run a number of games in which PAgent competes against other agents. The identities of these other agents and the resulting game logs are not made available to participants until after the competition. During the competition, participants connect to a game server which re-plays these games from the game logs. For each day of each game, participants receive the exact messages sent to PAgent (incoming messages), as well as the messages it sent to the game server in response (outgoing messages)–exactly the same information that would be available to an agent during a live game. In addition to these incoming and outgoing messages, each participant is also given a set of predictions that must be made before the information for the following day will be sent.

There are a number of benefits to running the competition using logs from completed games instead of using live games. First, there is no restriction on the number of participants that may compete head to head at one time. Second, each participant receives exactly the same information about the state of each game and is asked to make the same predictions. Finally, in live games there would be an incentive for participants to behave differently than in normal TAC SCM games, such as by manipulating prices in order to make past predictions come true.

Although predictions could be made on behalf of any agent from a completed game, the use of a single agent (PAgent) for which source code is available simplifies the task of participants by helping them to understand exactly what behavior to expect from the agent. PAgent was designed to be as simple as possible and to behave in a consistent and predictable manner while still exhibiting reasonable behavior. (We developed the PAgent as an extension of our TacTex Starter Agent[3], which is in turn a simplified version of TacTex made available for educational purposes.)

The exact predictions that are made by each participant are as follows:

- **Current computer prices:** The price at which each RFQ sent from customers on the current day will be ordered (i.e., the lowest price that will be offered by any manufacturer for that RFQ). These predictions are required on all but the first day and the last two days of each game, when few or no computers are sold. If the RFQ does not result in an order, the prediction will be ignored when accuracy is evaluated. Therefore, participants do not need to be concerned with whether an order will result, only what the price will be if there is an order.

- **Future computer prices:** For each of the 16 types of computers, the median price at which it will sell 20 days in the future. These predictions are required on all but the last 22 days of each game (thus the last day *on* which current computer price predictions are required is the last day *for* which future computer price predictions are required). If no computers of a certain type are sold, the prediction for that type will be ignored when accuracy is evaluated.

[3] `http://www.cs.utexas.edu/~TacTex/starterAgent`

- **Current component prices:** The price that will be offered for each RFQ sent by the PAgent to a supplier on the current day. The PAgent sends RFQs to suppliers on all but the last 10 days of each game. If an RFQ results in no offer (due to the reserve price) or an offer (or offers) with modified quantity or due date, the prediction for that RFQ will be ignored when accuracy is evaluated.

- **Future component prices:** The price that will be offered for each of a number of provided RFQs that will be sent by the PAgent to suppliers in 20 days. For each of the 16 pairs of a supplier and a component that it supplies, a zero-quantity RFQ is provided that will be sent by the PAgent in 20 days with a due date chosen at random between 5 and 30 (or the number of days remaining, if less than 30) days after the date the RFQ is sent. Because the PAgent sends no RFQs during the last 10 days of a game, predictions for future RFQs do not need to be made during the last 30 days of the game.

To test the ability of participants to make predictions for games with various agents, each participant is required to make predictions for 3 sets of games. In each set, the PAgent will have run against a different group of five competitors chosen at random from the TAC agent repository. Each set contains 16 games, meaning that participants have a chance to improve their predictions through repeated experience with the same group of competitors. Participants make predictions for one game at a time, and must complete the predictions for one game day before receiving information for the next day. Unlike the standard SCM game, participants do not need to compete simultaneously, so they may connect to the game server at any time and make predictions at their own pace. There is, however, an eight hour time limit.

Performance is evaluated separately for each of the four prediction categories. Root mean squared error is used as the scoring metric[4], and all errors are measured as a fraction of the base price of the computer/component. Participants are ranked in each category, and the overall winner is the agent with the highest average rank over all four categories.

5.3.2 Prediction Methods

Four participants competed in the 2007 Prediction Challenge: Botticelli (Brown University), DeepMaize (University of Michigan), Kshitij (Indian Institute of Technology Kharagpur), and TacTex. TacTex and DeepMaize have been among the top three agents in nearly every full SCM competition, and Botticelli has been a regular finalist. Here I describe the prediction methods used by the top two agents. The methods used by Botticelli and Kshitij have not been published or made known.

TacTex uses the price prediction methods used above, including the use of learning to adjust its future predictions, with a few small changes. During the full TAC SCM game, TacTex uses the distributions generated by its particle filters to estimate the probability of winning an order given a certain offer price. In the Prediction Challenge, for each current computer RFQ TacTex predicts that the sales price will be the mean of the distribution for that computer, or the price offered by the PAgent if that is lower. For predicting future computer prices, TacTex learns a model of price changes over 20 days, rather than the 10 days used previously, as this is what it is required to predict. For predicting future component prices, TacTex learns using the same features as before, but since only

[4] In general, RMS error is used as the loss function for all TAC-related predictions in this dissertation. While we are ultimately concerned with how these predictions affect agent performance, directly measuring this quantity is quite time consuming and is performed only in specific cases. The choice of RMS error is based on the fact that i) it is not apparent that errors in a particular direction are more harmful, at least in the aggregate, and ii) small errors are likely not serious, while a few large errors could potentially be catastrophic.

the changes in prices over 20 days are of interest, all training instances use only 20 as the number of days in which the RFQ will be sent. Training data is generated by playing a large number of games including a variety of agent groups from the TAC agent repository, with the PAgent included in each game.

DeepMaize [55] makes predictions for current and future computer prices using a k-nearest neighbors algorithm. For each prediction to be made, similar situations from a data set of previous games are identified, and the prediction is based on the prices observed in those situations. Predictions can be made about both the probability of winning an order at a given price and the expected winning price. Situations are chosen and weighted using Euclidean distance between a set of state features such as the date, estimated levels of supplier capacity and customer demand, and observed computer prices. Each neighbor is chosen from a different past game to provide sufficient diversity. DeepMaize uses two separate data sets, one from past TAC SCM tournament data and one from self-play, and updates the weighting of each set online based on past accuracy.

DeepMaize tracks component prices by recording the prices offered by each supplier over a number of recent days (five days in the full SCM competition, but only one day in the Prediction Challenge). The price for a component request with a given due date can then be predicted by taking the recorded price for that due date, if one exists, or by linearly interpolating between prices offered on different due dates if not. To improve the resulting predictions, DeepMaize also uses the reduced error pruning tree from WEKA, a form of regression tree, to learn the difference between actual observed prices in a data set of past games and the predictions of the linear interpolation method. This is essentially the same as the procedure used by TacTex to adjust future component price predictions.

Name	*Error*
1. TacTex	0.0455
2. DeepMaize	0.0468
3. Botticelli	0.0471
4. Kshitij	0.0487

Table 5.6: Current computer prices

Name	*Error*
1. TacTex	0.0916
2. DeepMaize	0.0959
3. Botticelli	0.1024
4. Kshitij	0.1109

Table 5.7: Future computer prices

Name	*Error*
1. DeepMaize	0.0392
2. Botticelli	0.0417
3. TacTex	0.0428
4. Kshitij	0.1333

Table 5.8: Current component prices

Name	*Error*
1. DeepMaize	0.0943
2. Botticelli	0.0970
3. TacTex	0.1034
4. Kshitij	0.1389

Table 5.9: Future component prices

5.3.3 2007 Results

Tables 5.6-5.9 show the prediction accuracy of each participant in each prediction category in terms of RMS error. Table 5.10 shows the overall place and average rank of each participant. Table 5.11 shows the five agents against which the PAgent competed in each of the three sets of 16 games. The winning participant, DeepMaize, had the lowest error on both current and future component price predictions, while TacTex had the lowest error on both current and future computer price predictions. For each category, the difference between the top agent and other agents is statistically significant with at least 98% confidence according to paired t-tests comparing the RMS errors for each of the 48 games. A few observations can be made from these results.

First, in each prediction category, the difference between the best and third best RMS error was fairly small, at most 12%. This fact suggests that the prediction methods used by the top three participants are all reasonably effective, and that there may be limited room for improvement. At the same time, the magnitudes of these errors are significant, suggesting that making predictions in TAC SCM is inherently difficult. To give perspective to these results, the agents in the final

Place	*Name*	*Avg. rank*
1	DeepMaize	1.5
2	TacTex	2
3	Botticelli	2.5
4	Kshitij	4

Table 5.10: Overall placing and average rank of each participant

Set	*Agents*
A	Maxon06, MinneTAC05, DeepMaize05, Foreseer05, PhantAgent06
B	GoBlueOval05, GeminiJK05, RationalSCM05, PhantAgent05, TacTex06
C	PhantAgent06, Maxon06, RationalSCM05, Tiancalli06, PhantAgent05

Table 5.11: Agents in each of the three sets of games

round of the 2007 TAC SCM competition had average profit margins between 1% and 7.5%, so prediction errors of these (similar) magnitudes could conceivably have a significant impact on agent performance.

Also, for both computers and component prices, the ranking of participants is the same for both current and future predictions. This is perhaps not surprising, as it seems reasonable that a participant able to make better short term predictions would have an advantage in making long term predictions. As expected, errors for future price predictions are much higher than errors for current price predictions, roughly by a factor of two.

5.3.4 Average Daily Errors

I begin my analysis by looking at how prediction errors vary across time. Figures 5.3-5.6 show the average RMS errors in each prediction category over all 48 games for each game day. (To improve visibility only the top three participants are shown; Kshitij's errors are consistently higher without displaying notably different patterns.) The most obvious feature of these graphs is that errors are usually very high at the beginning and end of games. Making predictions at the beginning of games can be difficult because there is little or no information about previous prices, and because prices can change rapidly as agents place large component orders (driving component prices up) and begin selling computers as components arrive (driving computer prices down). Computer prices are often unpredictable at the ends of games when agents are trying to sell off their remaining inventory–for each computer type, prices may suddenly become very high or low depending on inventory levels and thus competition. TacTex appears to suffer the most from errors at the start and end of games, especially when predicting component prices, while DeepMaize has particularly low errors in initial component price predictions and is roughly the same as Botticelli elsewhere. Occasional large errors such as these can be very damaging to a participant's overall performance due to the fact that RMS error, and not mean absolute error, is used in scoring.

In some cases, sudden error spikes can be attributed to the behavior of a specific agent. The spike in current component price errors on day 201 (and thus in future component price errors on day 181) occurs only in the games in Set A and is caused by MinneTAC05 sending large requests for components on that date but not accepting the resulting offers, presumably with the goal of driving up prices for other agents. It is interesting to note that Botticelli and TacTex recovered completely (returned to the previous low error level) in two days, while DeepMaize recovered in three days, suggesting that such spikes will only confuse agents for a short period of time.

The timing of the distinct jumps in late-game current computer price prediction errors observable in Figure 5.3 can also be traced to specific agents. The jump at day 202 occurs only in games from set C and is caused by Tiancalli06 suddenly dropping the prices it offers, while the jump at day 209 occurs only in games from set B and is caused by GeminiJK05 doing the same. The final rise over the last few days appears to be caused by widely varying (often very high) prices resulting from reduced competition to sell certain types of computers.

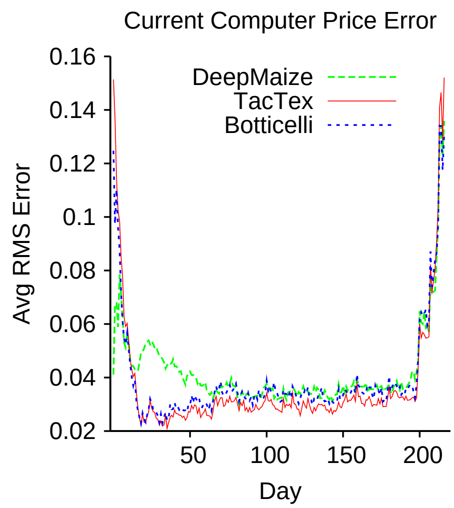

Figure 5.3: Current computer prices (avg. over all games)

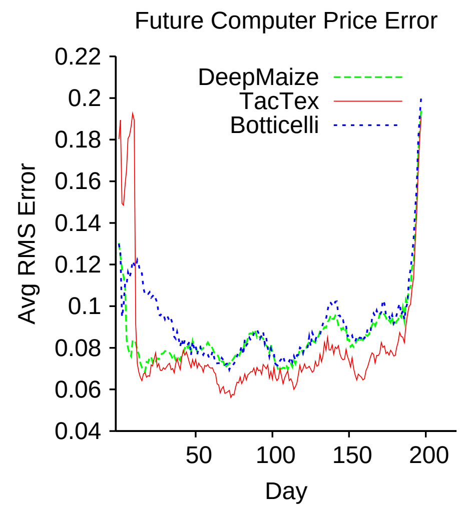

Figure 5.4: Future computer prices (avg. over all games)

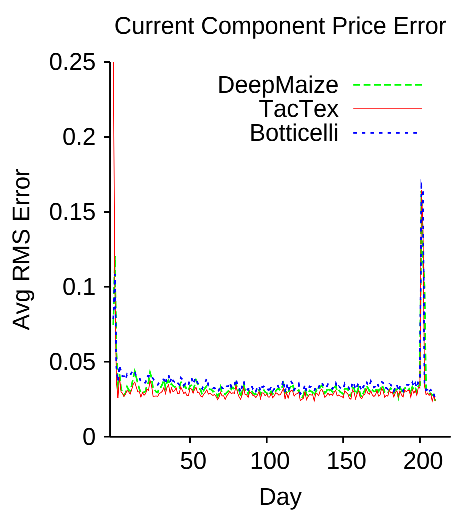

Figure 5.5: Current component prices (avg. over all games)

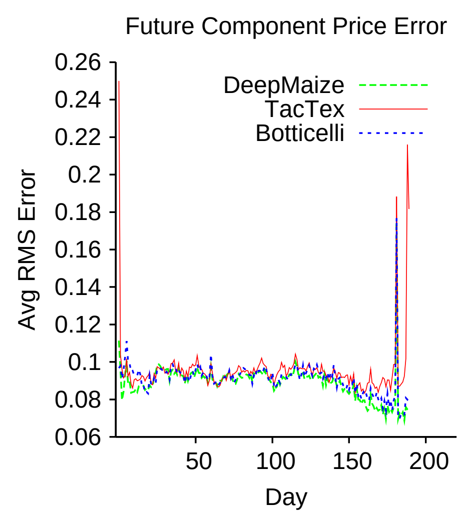

Figure 5.6: Future component prices (avg. over all games)

Compared to the starting and ending errors, average prediction errors during the middle of games tend to be much lower, and they are more consistent both over time and between participants. Still, there are some notable patterns. TacTex consistently has slightly lower errors for current computer price and current component price predictions and significantly lower errors for future computer price predictions, but errors for future component price predictions are generally a little higher than those of Botticelli or DeepMaize. DeepMaize suffers early on from higher errors for current computer price predictions, while Botticelli likewise has higher errors for future computer price predictions over the first portion of games, but otherwise the two participants have extremely similar patterns of errors.

The level of errors for current component prices remains nearly constant throughout games, while errors for future computer prices undergo notable swings for reasons that are unclear. These swings appear to some degree when each of the three sets of games is considered alone, although the swings occur at different times and scales for each set. While somewhat consistent, errors for current computer prices tend to be lower in the early parts of games for TacTex and Botticelli (probably due to the fact that competition tends to remain strong across all computer types while agents work through the components ordered at the start of each game), and errors for future component prices drop near the ends of games for Botticelli and DeepMaize (probably due to the fact that component orders, and thus changes in supplier prices, tend to dwindle during this period).

It is important to note that these observations do not necessarily hold when individual games are analyzed. Figures 5.7-5.10 show the daily RMS errors for a single representative game, game 3 from Set A. The most striking difference is the fact that for both current and future computer price predictions, errors vary considerably between the participants. Errors also show more variance across time, except for current component prices, where there are only occasional spikes that are likely caused by unusually heavy component requests.

In the remainder of this analysis, I will ignore errors over the first and last 20 days for which predictions are required for each prediction category. Doing so removes the highly variable effects (start and end game conditions, and the spike in component price errors caused by MinneTAC05) that can obscure patterns that

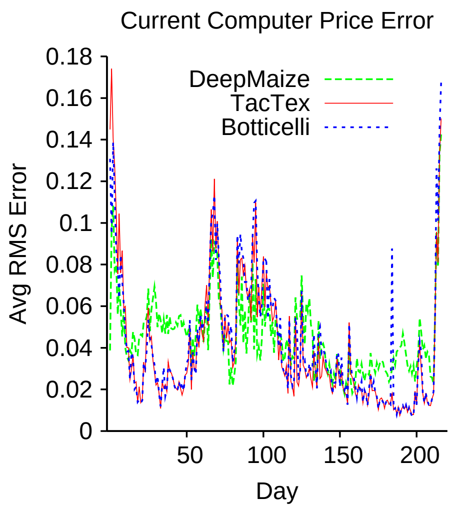

Figure 5.7: Current computer prices (game A-3)

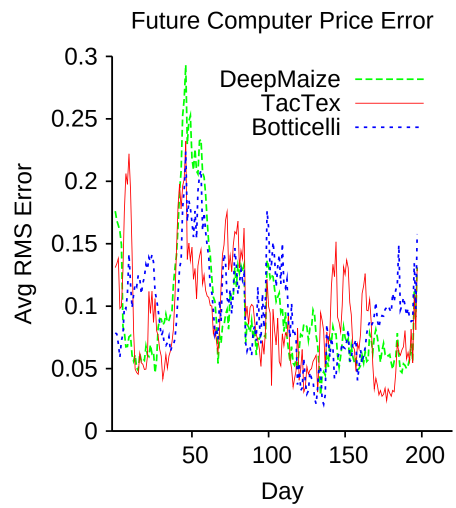

Figure 5.8: Future computer prices (game A-3)

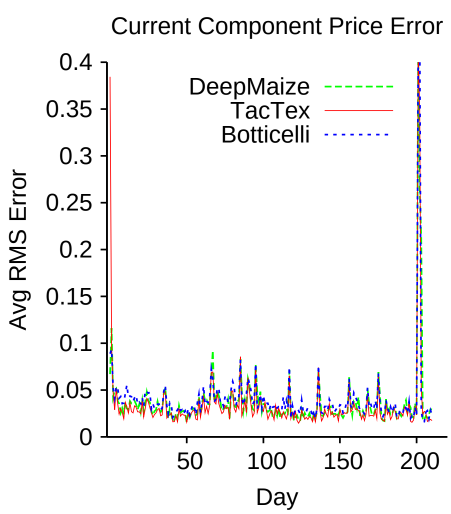

Figure 5.9: Current component prices (game A-3)

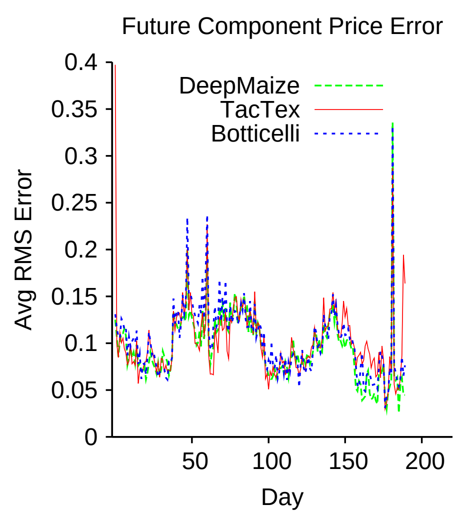

Figure 5.10: Future component prices (game A-3)

would otherwise be visible. Table 5.12 shows how the elimination of these errors affects the results.

Name	Current computer prices			Future computer prices		
	start	*mid*	*end*	*start*	*mid*	*end*
DeepMaize	**0.0562**	0.0403	**0.0947**	**0.0965**	0.0913	0.1356
TacTex	0.0771	**0.0342**	0.1026	0.1473	**0.0774**	**0.1262**
Botticelli	0.0665	0.0381	0.0984	0.1240	0.0952	0.1408

Name	Current component prices			Future component prices		
	start	*mid*	*end*	*start*	*mid*	*end*
DeepMaize	**0.0484**	0.0341	0.0810	**0.0920**	**0.0951**	**0.0936**
TacTex	0.0868	**0.0313**	**0.0797**	0.1219	0.0992	0.1210
Botticelli	0.0505	0.0365	0.0858	0.0965	0.0975	0.0969

Table 5.12: RMS errors over the first 20 days, last 20 days, and middle portion of the prediction interval for each category (lowest error in bold)

5.3.5 Differences Between Participants Across Games

To get a better view of how prediction error varies across games, I now compare the performance of participants on each game individually. Figures 5.11-5.14 show the errors for TacTex in each game plotted against those for DeepMaize. Each figure shows a different prediction category. Figures 5.15-5.18 show the same information for Botticelli and DeepMaize, and Figures 5.19-5.22 compare TacTex to Botticelli. For each figure, the correlation coefficient r is given. The dotted line in each figure is the line $y = x$, meaning that a point below the line represents a game for which the participant on the y-axis had lower error.

I begin by looking at the current computer price predictions. Figure 5.19 shows that the errors of TacTex and Botticelli are highly correlated, with Botticelli's errors being higher than TacTex's by a similar amount in each game. Comparing either participant to DeepMaize (Figures 5.11 and 5.15) paints a different picture. While there is still a strong correlation between errors, it appears that as the difficulty of making predictions in a game increases, the performance of DeepMaize increases relative to the performance of the others, to the point that DeepMaize has the lowest errors of any participant on the most difficult games. One possible

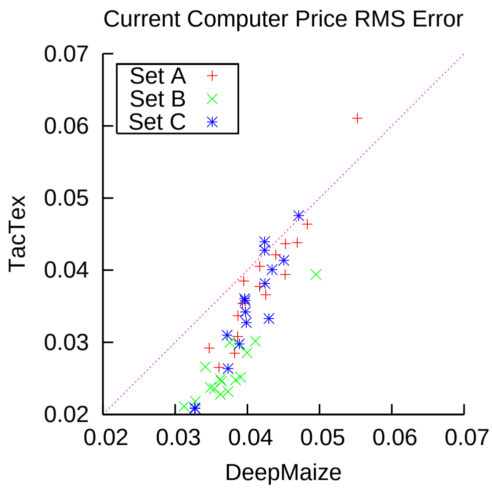

Figure 5.11: Current computer prices (each game, TT / DM, r=.92)

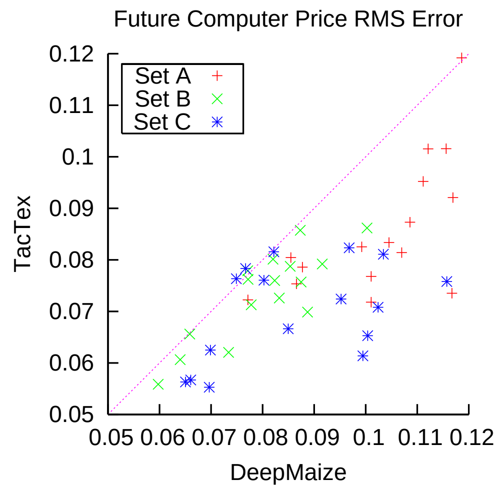

Figure 5.12: Future computer prices (each game, TT / DM, r=.70)

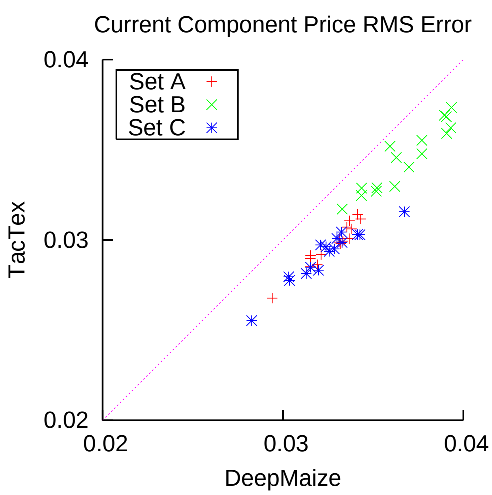

Figure 5.13: Current component prices (each game, TT / DM, r=.97)

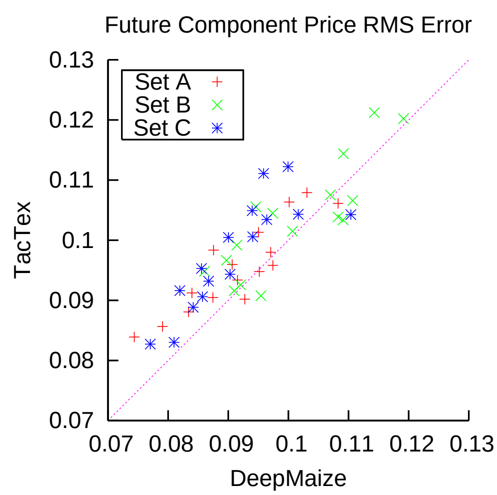

Figure 5.14: Future component prices (each game, TT / DM, r=.87)

explanation for this result is that the prediction methods of other participants (the particle filter in the case of TacTex) are highly tuned for "typical" games and thus suffer as computer prices behave more atypically, while DeepMaize's use of a kNN-based predictor allows it to better handle unusual situations by matching them with similar situations from its data set. This prediction category is the only one in

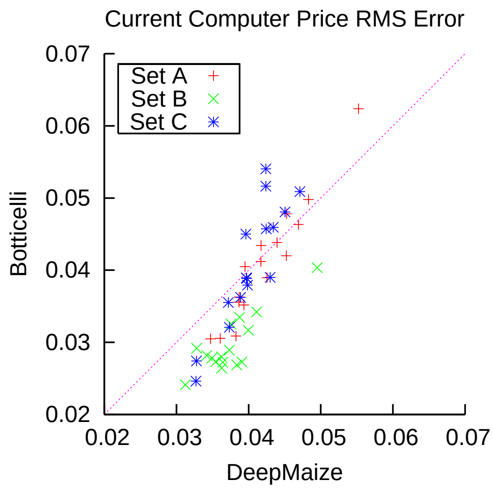

Figure 5.15: Current computer prices (each game, Bot. / DM, r=.87)

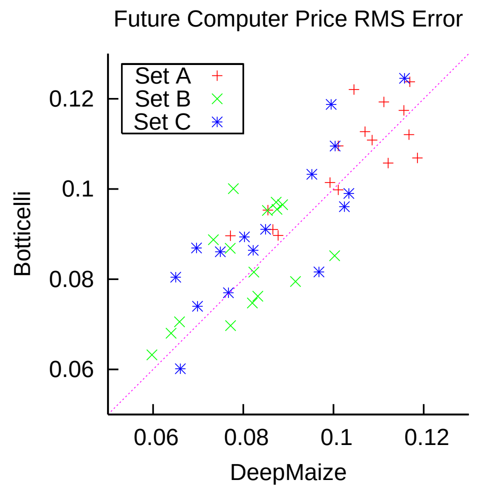

Figure 5.16: Future computer prices (each game, Bot. / DM, r=.85)

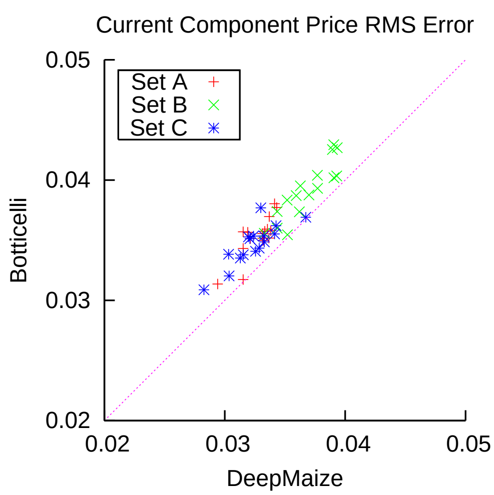

Figure 5.17: Current component prices (each game, Bot. / DM, r=.93)

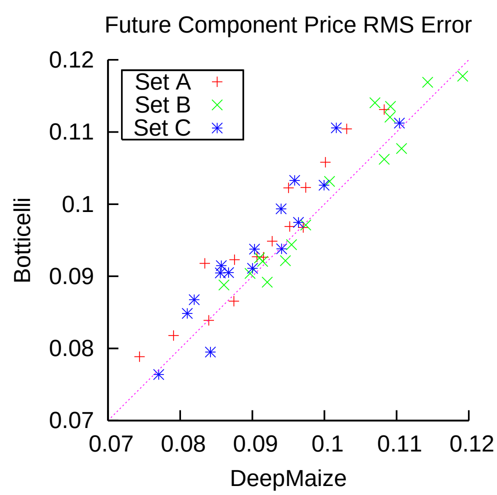

Figure 5.18: Future component prices (each game, Bot. / DM, r=.95)

which such a phenomenon occurs, and this fact is particularly interesting because it makes it difficult to state that one participant's method of prediction is best (in expectation) under all circumstances. An agent with access to the prediction methods of all participants might choose to use TacTex's method in most cases but to use DeepMaize's method in certain games where prediction appeared particularly

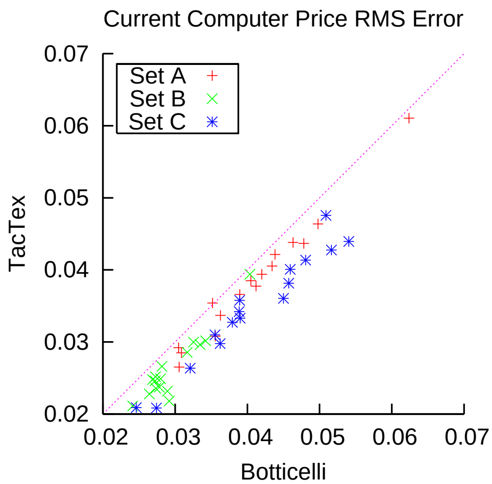

Figure 5.19: Current computer prices (each game, TT / Bot., r=.97)

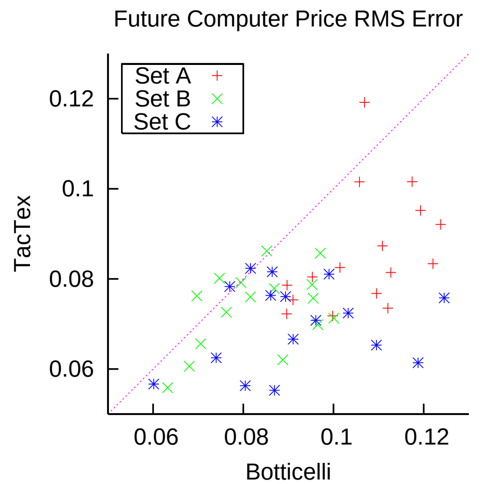

Figure 5.20: Future computer prices (each game, TT / Bot., r=.49)

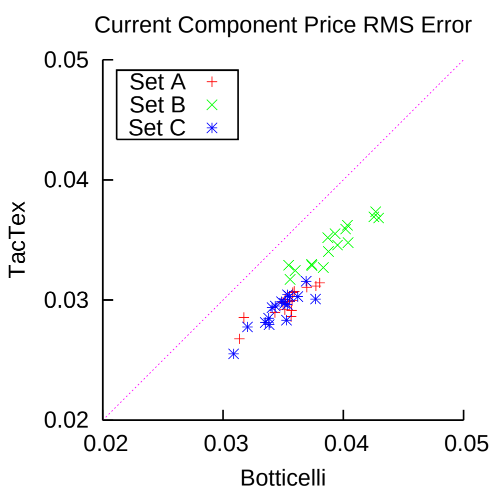

Figure 5.21: Current component prices (each game, TT / Bot., r=.94)

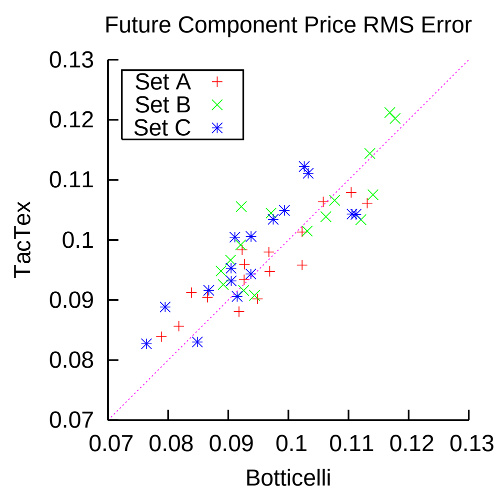

Figure 5.22: Future component prices (each game, TT / Bot., r=.87)

difficult.

Errors for future computer prices (Figures 5.12, 5.16, and 5.20) exhibit a different pattern. Here there is some correlation between the errors of DeepMaize and Botticelli, but very little between the errors of either of these two participants and those of TacTex. In fact, for Set C, the errors of TacTex appear completely unrelated to those of the other two participants. This low correlation suggests that the difficulties experienced by DeepMaize and Botticelli are not related to a particular set of games or common to all games, but have to do with particular situations that can occur in all three sets of games and that TacTex is able to handle correctly. In the case of DeepMaize, these situations may be different from those encountered in the data set used by the kNN-based predictor, or the distance metric used by the predictor may be unable to distinguish these situations from unrelated ones in the data set. The fact that DeepMaize and Botticelli have a higher degree of correlation suggests that they may have difficulties under some of the same circumstances.

The pattern of errors for current component prices (Figures 5.13, 5.17, and 5.21) is much clearer. Here there is a high degree of correlation between the errors of different participants, with the errors of one participant differing from the errors of another by a fairly consistent amount. It should be noted that the reason why TacTex has the lowest errors in these figures, but the third lowest error in Table 5.8, is the exclusion of the beginning of each game, where TacTex had very high errors.

While not as highly correlated as the errors in current component prices, the errors for future component price predictions (Figures 5.14, 5.18, and 5.22) show a somewhat similar pattern.

In addition to making comparisons between the participants, we can also compare the difficulty of making predictions for each of the three sets of games. For computer prices, it appears to be easier to make predictions for Set B, especially current predictions, while Set A tends to have higher future prediction errors. On the other hand, predicting component prices appears to be more difficult for Set B, especially current component prices. The reasons for these differences between sets are not clear, unlike the error spikes in Figures 5.3 and 5.5 that could be traced to specific agent behaviors. Better understanding these differences would likely be useful in designing improved predictors that can handle a wider variety of agent behaviors.

It is interesting to note that the patterns observed above (such as correlations between errors and which participant had the lowest errors) generally appear to hold equally well for all three sets of games. Given that the prediction methods used often require the developer to choose a training data set composed of past game results, it would not be surprising for a participant to make particularly accurate predictions on games that are most similar to the games in the chosen data set, and for certain participants to favor certain sets of games as a result, but this does not appear to have happened.

5.3.6 Differences Between Participants Across Days

To make comparisons at a finer level of detail, the errors of each agent can also be plotted on a daily basis, rather than for each game. Figures 5.23-5.28 show a subset of the comparisons from Figures 5.11-5.22 at this level. Again, RMS error is measured, and the first and last 20 days of errors are omitted. These figures largely serve to shed further light on the observations that have been made previously. Unlike the previous set of figures, no indication is given about the set of games from which each point plotted came, but plotting each set separately reveals very similarly shaped distributions for each set.

Figure 5.25 shows that the daily errors for the current computer predictions of TacTex and Botticelli are highly correlated, as would be expected from Figure 5.19. The correlation between TacTex and DeepMaize is much weaker, as seen in Figure 5.23 (the plot of Botticelli and DeepMaize is nearly the same). As noted before, Figures 5.11 and 5.15 show that the performance of DeepMaize tends to improve relative to the other participants as the predictions become more challenging. While a distribution of the same shape (high correlation but a high slope)

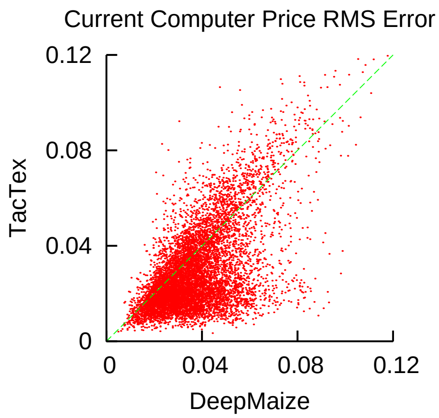

Figure 5.23: Current computer prices (each day, TT / DM, r=.60)

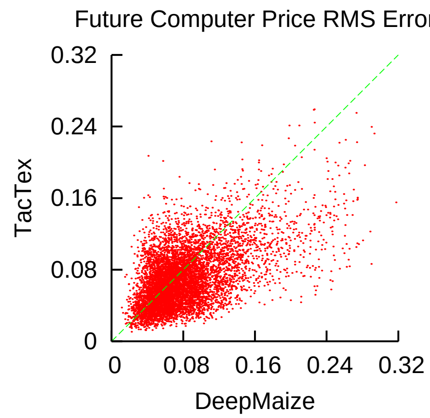

Figure 5.24: Future computer prices (each day, TT / DM, r=.55)

in Figure 5.25 would cause this outcome, instead it appears that there are some predictions for which DeepMaize has similar errors (those along the line $y = x$), along with a cluster of predictions (along the bottom-left) for which DeepMaize has higher errors. It may be the case that there are certain relatively easy predictions with which DeepMaize has difficulty, and that these easy predictions occur less often in the more challenging games. Many of the points in this cluster are from the early parts of games where DeepMaize has higher errors (see Figure 5.3), but not all–even with the first 70 days omitted from the plot, the cluster is still visible.

Based on Figures 5.12 and 5.16, we would expect DeepMaize's daily future computer price prediction errors to be weakly correlated with those of TacTex and somewhat correlated with those of Botticelli, and Figures 5.24 and 5.26 confirm this expectation (the plot of TacTex and Botticelli is similar to Figure 5.24). Looking at Figure 5.4, in which Botticelli and DeepMaize have nearly identical average daily errors, it is perhaps surprising that their correlation here is not higher.

Figures 5.27 and 5.28 show the daily errors for the current and future component price predictions of DeepMaize and TacTex (plots for Botticelli are similar). These errors are highly correlated, as they were in Figures 5.13 and 5.14. Figure 5.27 illustrates that the pattern observed for a single game in Figure 5.9 (mostly

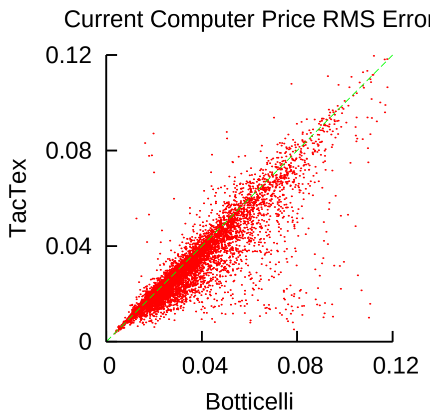

Figure 5.25: Current computer prices (each day, TT / Bot., r=.89)

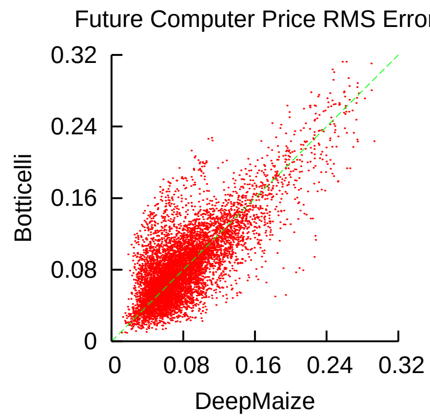

Figure 5.26: Future computer prices (each day, Bot. / DM, r=.77)

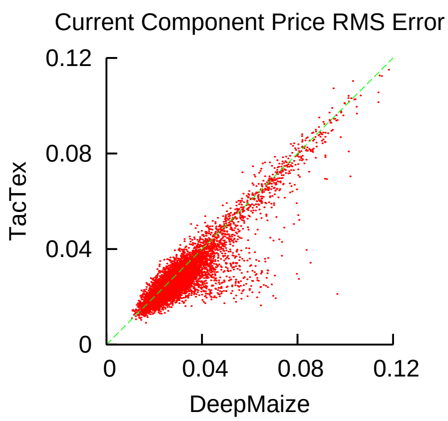

Figure 5.27: Current component prices (each day, TT / DM, r=.90)

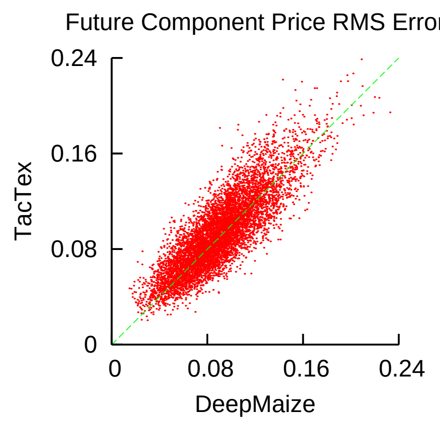

Figure 5.28: Future component prices (each day, TT / DM, r=.85)

low errors around 0.03, with occasional spikes that affect all participants similarly) is true in general. Similarly, the pattern seen in one game in Figure 5.10 (errors more evenly distributed over a wide range but still highly correlated between participants) appears in Figure 5.28.

One additional observation that can be made is that while a participant may show consistently lower errors at the full-game level (for instance, TacTex in Figures 5.12 and 5.13), there may still be a large number of days on which it has higher errors (Figures 5.24 and 5.27). This observation may indicate that there

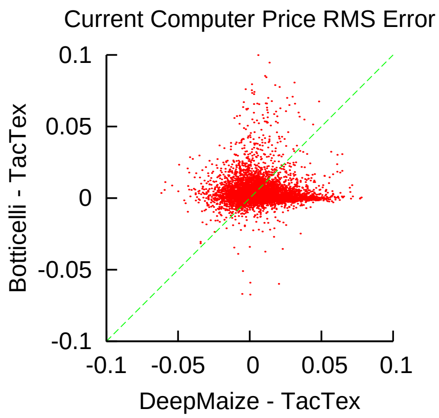

Figure 5.29: Current computer prices (daily differences with TT, r=-.05)

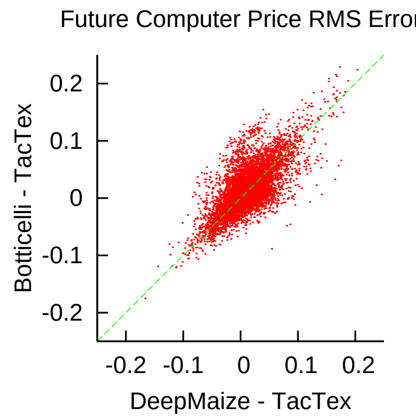

Figure 5.30: Future computer prices (daily differences with TT, r=.71)

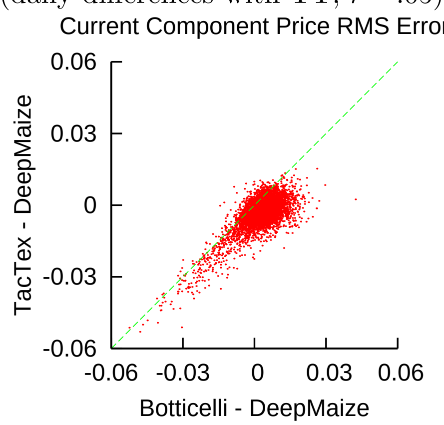

Figure 5.31: Current component prices (daily differences with DM, r=.70)

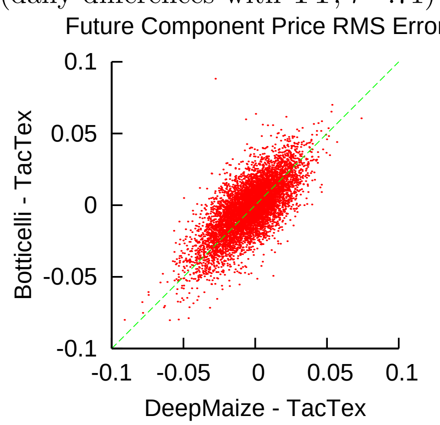

Figure 5.32: Future component prices (daily differences with TT, r=.72)

is still room for the participant to improve, or it may simply be a result of the stochastic nature of the game (that is, for each situation in which predictions are made, there may actually be a wide distribution over possible outcomes depending on random game factors such as demand fluctuations). Both possibilities are likely true to some degree.

An alternative way of presenting this data is to look at the *differences* between the errors of participants and to plot these differences for different pairs of participants. For example, in Figure 5.32 the x-axis shows the daily future component price prediction errors of DeepMaize minus those of TacTex, while the y-axis shows the errors of Botticelli minus those of TacTex. In this case, the points show a moderately high degree of correlation ($r = .72$), indicating that the situations in which TacTex differs in accuracy from DeepMaize tend to be the same as those in which it differs from Botticelli. In most cases, these plots are fairly uniform clusters with low correlation, but there are some exceptions, as shown in Figures 5.29-5.32. Figure 5.29 shows that for current computer price predictions, DeepMaize tends to vary more widely from the accuracy of TacTex than Botticelli does. Interestingly, for those situations in which DeepMaize has considerably higher errors than TacTex (those extending to the right here, and seen on the bottom-left of Figure 5.23), TacTex and Botticelli have nearly identical errors. For current component price predictions, Figure 5.31 shows that TacTex and Botticelli usually differ only slightly from DeepMaize and in a weakly correlated way, but that there are certain situations in which DeepMaize has much higher errors than them both. Figures 5.30 and 5.32 show that DeepMaize and Botticelli differ from TacTex in similar situations for future computer and component price predictions. The degree of correlation for these two categories drops considerably when comparing differences with a participant other than TacTex, suggesting that Botticelli's methods of making predictions of future prices have more in common with those of DeepMaize than TacTex.

5.3.7 Error Persistence

Another question that can be addressed is whether errors tend to persist. That is, if a participant has a high prediction error on one day, will it also have a high error on the next day? We answer this question by plotting daily errors against the next day's errors. In most cases, the answer is yes; for current and future computer price predictions and future component price predictions, errors are highly correlated across days, with correlation coefficients above 0.73 for current

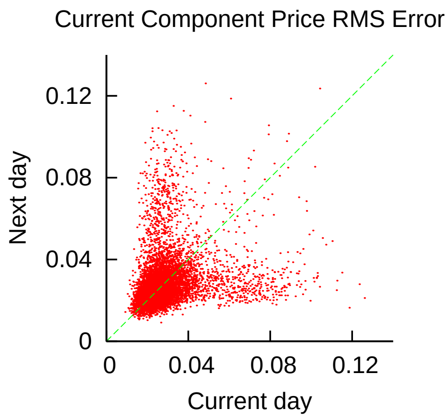

Figure 5.33: Current component prices
(errors on consecutive days, TT, r=.23)

computer price predictions and above 0.82 for both future price predictions for each participant. The fact that correlation is highest for the future price predictions makes sense given that game conditions can change significantly over 20 days while the information available to make predictions about these conditions changes little from day to day. In the case of current component price predictions, however, there is little correlation between daily errors. Figure 5.33 shows the plot for daily errors of TacTex on consecutive days, and the plots for other participants are similar. There appear to be a large number of cases in which the error jumps greatly for a single day (the vertical arm) and then decreases to a more normal level on the following day (the horizontal arm). Even on days with more moderate errors, the degree of correlation between days is fairly low. These observations make sense given the error pattern seen in Figure 5.9.

As before, we can also consider the differences between the error rates of different participants. When daily differences in errors are plotted for consecutive days, the differences in future computer price prediction errors turn out to be highly correlated across days ($r > 0.85$ for each pair of participants), and the differences in future component price prediction errors turn out to be moderately correlated (r of about 0.6 for each pair). For current computer price predictions, the results depend

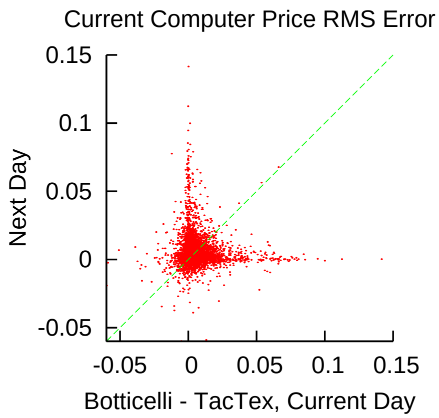

Figure 5.34: Current computer prices (daily differences, Bot-TT, r=.03)

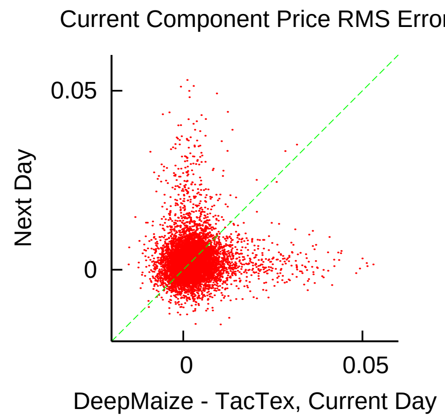

Figure 5.35: Current component prices (daily differences, DM-TT, r=.71)

on the pair of participants considered. While the differences between the errors of DeepMaize and TacTex show a moderately high degree of correlation across days ($r = 0.77$), the differences between Botticelli and TacTex are not only uncorrelated across days, but as shown in Figure 5.34, there are a number of cases in which the error of Botticelli jumps from being nearly the same as TacTex to being much higher for a single day. For current component price predictions, a similar pattern emerges when comparing DeepMaize to either TacTex (Figure 5.35) or Botticelli, while the differences between the errors of TacTex and Botticelli are similarly uncorrelated across days without showing such jumps.

5.3.8 Individual Error Distribution

Finally, it is possible to look beyond the RMS error of a set of predictions covering a whole game or day and consider prediction errors for individual predictions. As this level of detail is not available in the competition logs, I give here only a brief look at the data we were able to generate for TacTex after the competition. Instead of considering RMS error, I look at the actual errors to determine their distribution. For current price predictions, errors are grouped into 40 bins of width 0.005 spanning the range [-0.1, 0.1], and for future price predictions, errors are grouped into 40 bins of width 0.015 spanning the range [-0.3, 0.3]. Errors falling above or below these ranges are grouped into an additional bin. Figures 5.36-5.39 show the resulting histograms. In each category, errors appear to be nearly normally distributed with a mean near zero. This result is not necessarily unexpected, as we would expect a normal distribution if the errors were due to a large number of uncorrelated factors. However, given the nature of the TAC SCM game, it would also not have been surprising if a more interesting pattern had emerged. For instance, it is fairly simple for a single agent to drive computer prices down by offering lower prices, or to drive component prices up by requesting a large number of components. It is less likely for these prices to suddenly move in the opposite direction, and so the distributions could conceivably have shown a tendency toward larger errors in one direction. The fact that this tendency was not observed suggests that the prediction methods used may already account for such factors.

5.3.9 2008 Results

The results of the 2007 Prediction Challenge showed that TacTex's performance was very strong overall, with the glaring exception of the beginnings and endings of games. In the full SCM game, TacTex handles these situations as special cases and generally does not make use of its usual price predictions. In fact, when generating training data for TacTex's models, we exclude a few days from the start and end of each game for this reason. To address TacTex's flaws in the Prediction Challenge, we could have either included these days in training or trained models specifically for these days, but instead we simply had TacTex submit hard-coded price predictions representing the average prices seen in the training data. This change was enough for TacTex to win the 2008 Prediction Challenge. Results are shown in Tables 5.13-5.17, and the groups of agents are shown in Table 5.18. TacTex extended its winning margin over 2007 in the current and future computer price categories, and was very competitive in the current and future component

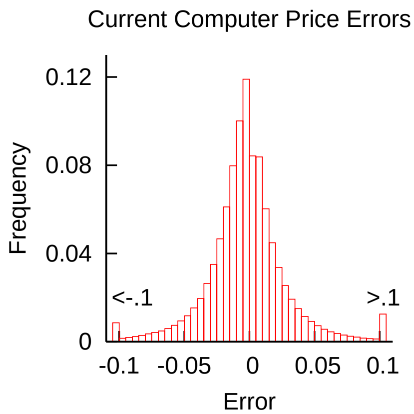

Figure 5.36: Current computer price error distribution for TacTex

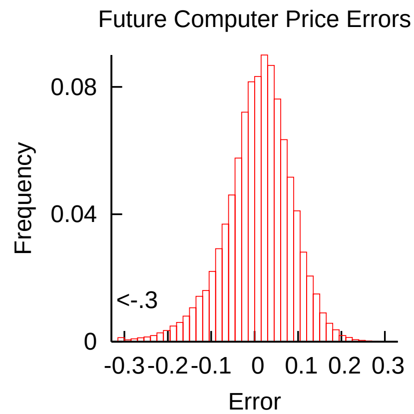

Figure 5.37: Future computer price error distribution for TacTex

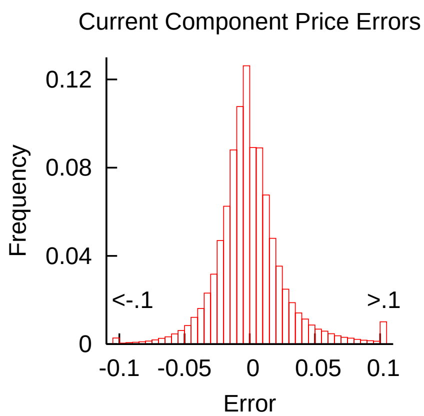

Figure 5.38: Current component price error distribution for TacTex

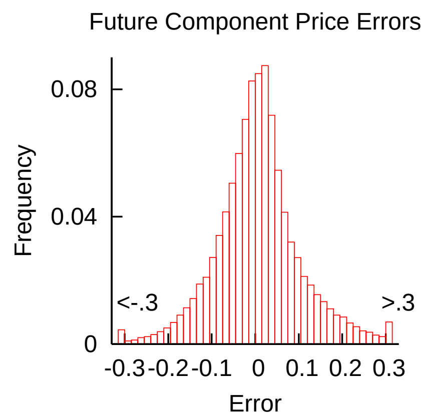

Figure 5.39: Future component price error distribution for TacTex

categories.

5.4 Learning Across Markets

Sections 5.1 and 5.2 showed the importance of learning to predict future price changes. The previous section showed that TacTex's predictions are reasonably accurate when compared to the predictions of other agents (and much more accurate in the case of future computer price predictions). As a result, it is quite likely that there is little room for improvement through modifying the approach

Name	*Error*
1. TacTex	0.0478
2. DeepMaize	0.0513
3. Botticelli	0.0531

Table 5.13: Current computer prices

Name	*Error*
1. TacTex	0.0873
2. DeepMaize	0.0975
3. Botticelli	0.0993

Table 5.14: Future computer prices

Name	*Error*
1. DeepMaize	0.0347
2. TacTex	0.0351
3. Botticelli	0.0403

Table 5.15: Current component prices

Name	*Error*
1. Botticelli	0.0996
2. TacTex	0.1023
3. DeepMaize	0.1028

Table 5.16: Future component prices

Place	*Name*	*Avg. rank*
1	TacTex	1.5
2	DeepMaize	2
3	Botticelli	2.5

Table 5.17: Overall placing and average rank of each participant, 2008

Set	*Agents*
A	PhantAgent05, GoBlueOval05, Mertacor05, Southampton05, Southampton05
B	GeminiJK05, GeminiJK05, TacTex05, Mertacor05, PhantAgent07
C	RationalSCM05, PhantAgent06, DeepMaize07, Foreseer05, Botticelli05

Table 5.18: Agents in each of the three sets of games, 2008

to learning, whether by trying additional learning algorithms or adding additional data features. However, one issue that deserves further consideration is the source of data used to train TacTex's models.

In Section 1.1 of the introduction, I outlined general approaches to adaptation that could be taken when previous market experience–in this scenario, completed games from either competition or our own simulations–is available. As described previously, in both the Prediction Challenge and the full SCM game, TacTex uses fixed models trained on games including a wide variety of participating agents. This approach amounts to learning from previous experience only, and so in this section, I analyze how well models trained on one group of agents generalize to a different group. Another approach would be to learn online from only the new market (i.e., current round of competition), and so I also evaluate the effectiveness this approach.

5.4.1 Generalization

I begin by defining a number of agent groups between which comparisons will be made. The first four groups contain a variety of agents from the agent repository:

- R1: TacTex-06, GeminiJK-05, Mertacor-05, MinneTAC-06, PhantAgent-06, RationalAgent-05
- R2: TacTex-06, TacTex-05, Botticelli-05, CrocodileAgent-05, DeepMaize-05, GoBlueOval-05
- R3: TacTex-06, DeepMaize-06, Foreseer-05, Maxon-06, MinneTAC-05, PhantAgent-05
- R4: TacTex-06, TacTex-05, DeepMaize-06, Maxon-06, MinneTAC-06, PhantAgent-06

Group R4 contains the five agents from the 2006 final round for which binaries are available, along with TacTex-05, and is thus a much stronger group of agents than groups R1-R3. In these games, TacTex-06 used the computer price model that was used in the 2006 competition. Recall that TacTex-06 did not make use of a component price model, and TacTex-05 did not use either model. The combined data from these four groups was used to train the models that were used in the 2007 through 2009 versions of TacTex, and I will refer to these models as the *default* models.

The next group consists of the the actual data from the 2010 final round:

- F10: TacTex-10, DeepMaize-10, MinneTAC-10, Botticelli-10, Nanda-10, Mertacor-10

TacTex-10 used computer and component models trained on the semifinal round of the 2010 competition.

The next group includes five agents that were based on the TacTex Starter Agent and developed by students for a class in 2006. As a result of the limited development time that was available, these agents generally exhibit simple behavior and are not as strong as most of the agents in other groups.

- C06: TacTex-10, Simplicity, Redbull, StormFront, Garfield, JAgent

The final three groups consist of two copies of TacTex-10 using the default models (labeled TacTex-10') against four copies of one agent: TacTex-05, TacTex10 (using the models used in competition), or DeepMaize-09 (the 2009 winner).

- TT05: TacTex-10' * 2, TacTex-05 * 4
- TT10: TacTex-10' * 2, TacTex-10 * 4
- DM09: TacTex-10' * 2, DeepMaize-09 * 4

At least one version of TacTex is present in each group because we are only interested in making predictions for games in which we are playing, and our own actions influence the prices we are trying to predict. For each group, we ran 30 games, except for the 2010 finals which consisted of 36 games. To compare model generalization between groups, we trained one of each model on each group, using the learning algorithms chosen previously (additive regression for computers and M5P trees for components). In each case, we trained the type of model used by

	Test Data								
Model	R1	R2	R3	R4	F10	C06	TT05	TT10	DM09
R1	**.0613**	.0733	.0663	.0665	.0645	.0839	.0806	.0995	.0802
R2	.0646	**.0710**	.0676	.0671	.0627	.0832	.0791	.1008	.0775
R3	.0662	.0758	**.0611**	.0644	.0649	.0869	.0846	.1032	.0778
R4	.0668	.0760	.0645	**.0608**	.0642	.0885	.0829	.1037	.0810
F10	.0671	.0754	.0670	.0669	**.0580**	.0858	.0815	.1061	.0772
C06	.0679	.0749	.0713	.0743	.0681	**.0793**	.0795	.1018	.0790
TT05	.0697	.0798	.0756	.0733	.0704	.0871	**.0729**	.0996	.0840
TT10	.0733	.0843	.0774	.0767	.0750	.0907	.0806	**.0924**	.0827
DM09	.0687	.0785	.0668	.0689	.0635	.0852	.0829	.1003	**.0724**

Table 5.19: RMS error when predictive models are learned using games from one group and tested on games from another group - computer costs

TacTex for full SCM games (predicting 10 day price changes for computers, and 5-40 day price changes for components) rather than the Prediction Challenge (20 day price changes in each case). We then evaluated each model on all groups. For determining the error when a model was trained and tested on the same group, we used leave-one-game-out cross validation. The results are shown in Tables 5.19 (computers) and 5.20 (components). The lowest error on each group is shown in bold, and was statistically significant in each case with 95% confidence according to paired t-tests (comparing errors on individual games).

A number of observations can be made from these tables. As expected, for each group, the model trained on that group had the lowest error of any model. This observation confirms our assumption that there is an observable difference in behavior between different groups. In some cases, however, models trained on a different group had only slightly higher errors, suggesting that the differences in behavior were small. In addition, some groups present more difficult prediction problems than others. That is, model errors are higher on those groups than others, even for the models trained specifically on those groups. (This difference between groups was also apparent in the results of the Prediction Challenge.) For computer

	Test Data								
Model	R1	R2	R3	R4	F10	C06	TT05	TT10	DM09
R1	**.0467**	.0856	.0663	.0518	.0581	.1073	.0897	.0803	.0605
R2	.0646	**.0758**	.0789	.0661	.0706	.1076	.0982	.0875	.0735
R3	.0579	.0909	**.0554**	.0601	.0626	.1079	.0860	.0809	.0654
R4	.0500	.0845	.0654	**.0502**	.0578	.1042	.0847	.0781	.0608
F10	.0500	.0861	.0646	.0542	**.0523**	.1076	.0819	.0775	.0593
C06	.0939	.1000	.0999	.0882	.0970	**.0813**	.1102	.1052	.0969
TT05	.0586	.0928	.0710	.0610	.0667	.1093	**.0719**	.0817	.0672
TT10	.0597	.0908	.0705	.0609	.0629	.1073	.0853	**.0711**	.0634
DM09	.0523	.0859	.0660	.0547	.0558	.1061	.0823	.0753	**.0581**

Table 5.20: RMS error when predictive models are learned using games from one group and tested on games from another group - component costs

prices, the most unpredictable group was TT10, which consists of six copies of TacTex-10. This result is not particularly surprising, as TacTex is the agent that appears to react most strongly to anticipated changes in demand by building or liquidating its computer inventory (as discussed in Section 5.1), and six agents behaving this way can result in erratic prices. For component prices, the most unpredictable group was C06. These simple agents tend to exhibit little long-term planning, and so short-term component prices can be quite erratic. Finally, it is interesting to compare the differences in errors between models when tested on a particular group to the differences in errors between participants in the Prediction Challenge. Again, these errors are not directly comparable due to the fact that predictions in the Prediction Challenge are made for different time periods, but in many cases the differences in errors between different models are larger than the differences in errors between participants on a particular category of future price predictions. In particular, the errors of the 2008 participants on future component price predictions were within 0.0032 of each other, while the difference between model errors in Table 5.20 typically exceeds 0.01. It is thus possible that the outcome of the Prediction Challenge had as much to do with the choice of training data used by participants as with the agent design or any learning algorithms used.

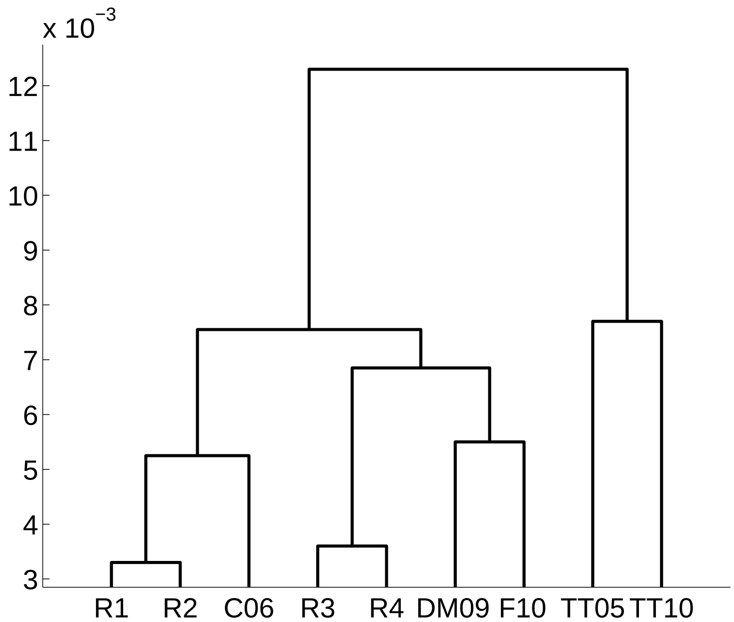

Figure 5.40: Dendrogram of computer cost similarity based on Table 5.19

As a way of better visualizing Tables 5.19 and 5.20, we developed a simple distance function for groups and performed clustering. Let $E(a, b)$ be the error of the model trained on group a when tested on group b, and let $D(a, b) = E(a, b) - E(b, b)$, so that D measures how much worse the model of a is than the model of b when tested on b. We define the distance $dist(a, b)$ to be $max(D(a, b), D(b, a))$, that is, the larger of these differences. (Taking the average produced similar clustering results.) Applying this distance function and performing agglomerative (bottom-up) clustering results in the dendrograms shown in Figures 5.40 and 5.41. The interpretation of these dendrograms is that each item (agent group) is joined to the nearest item or cluster of items (based on average distance), and the height of the line joining clusters represents the distance between them. Again, we see that some groups are fairly similar (DM09 and F10 are similar in both computer and component prices), while other groups are clear outliers (TT05 and TT10 for computer prices and C06 for component prices).

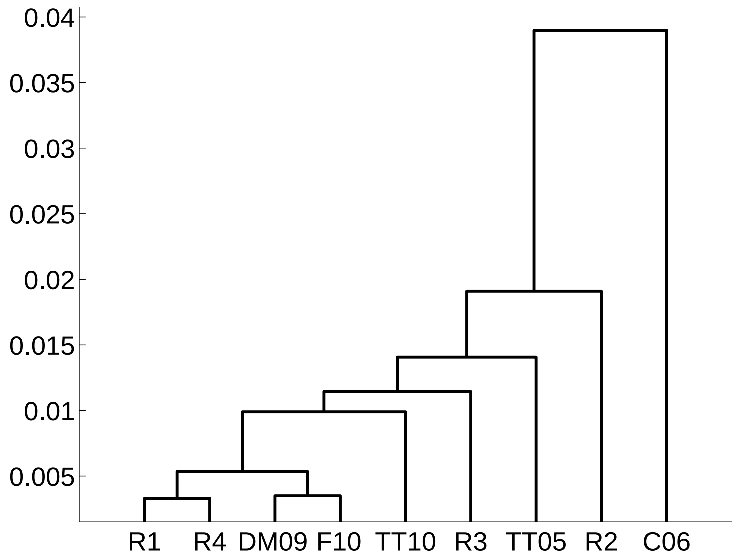

Figure 5.41: Dendrogram of component cost similarity based on Table 5.20

The natural question to ask next is whether prediction accuracy affects agent performance. We have already seen that predicting changes in computer and component prices improves performance over making no such predictions, but it is not obvious that small changes in prediction accuracy would have a similar impact. For instance, it could be the case that it is more important to predict the direction of price changes than the magnitude. Additionally, using different models will cause the behavior of TacTex to change and may alter the patterns of the prices we are trying to predict. (We ultimately view consideration of this issue to be the responsibility of the agent, and not the learning process–an agent should be able to account for the fact that by behaving as its models suggests it should, it may be affecting the economy in a way that makes its predictions incorrect.)

To test the importance of model accuracy, we ran experiments using the agents of groups TT10 and DM09, except that in each experiment one of the agents that had previously used the default models had one of its models (computer or component) changed to the model trained on that group. 30 games were run for

	Change computer model from default to group-trained	*Change component model from default to group-trained*
TT10	**+0.80**	**+0.26**
DM09	**+0.99**	-0.03

Table 5.21: Impact of model choice on agent performance. Differences in scores (in millions) between an agent using default models and an agent using the model shown.

each experiment. The differences in score between the changed agent and the agent using the default models are shown in Table 5.21. Differences that are statistically significant with 95% confidence according to paired t-tests are shown in bold. The results show that the accuracy of the computer model is particularly important, as using the model trained on a group increased the agent's score by almost one million over the agent using the default model. The importance of the accuracy of the component model is less clear. For the group TT10, there was a modest but statistically significant increase in score from using the more accurate model, while there was essentially no change for the group DM09.

5.4.2 Learning Online

The previous results show that using a model trained on the same group of agents against which TacTex is competing can be beneficial, especially in the case of the computer model. However, in a TAC SCM tournament, there is generally little opportunity to obtain training data for the group of agents competing in a given round before that round begins. (The same six agents compete in a number of games in each round.) While agent binaries from previous years may be available, agents may change significantly from one year to the next. Previous rounds of the tournament generally involve different agent groupings, and agents may change between rounds.

An alternative to using fixed models trained before a round is to train models online during the course of a round. There are two problems with this approach that should be acknowledged. First, there is only limited time between games, and so, depending on the learning algorithm used, there may be a delay between the time data becomes available and the time a new model trained on that data can be used. Second, since 2007 agents have not been allowed to access the game logs generated during the current round of competition. In all of the learning described so far, we have used these logs to generate training data. However, it is possible to generate this data using only the information revealed to TacTex during a game. Recall that only the label of each training instance uses information about true future prices, while the features all necessarily represent information known to TacTex at the time the prediction is needed. In the case of component prices, we know the exact prices offered by suppliers and so can still obtain true labels. For computer prices, we need to know the average price of each computer type each day. TacTex only observes the high and low price for each computer type as well as the acceptance of its own offers. As shown in Figure 5.1, however, the range of prices is often quite narrow, especially relative to the change in prices over 10 days, and so TacTex could obtain fairly accurate estimates of the changes in average prices. Nevertheless, for the sake of simplicity, in the experiments below I will assume that TacTex does have access to the game logs immediately following each game and that it can train new models before the next game begins.

The main potential concern with learning online is the fact that the limited amount of data might not be enough to learn accurate models. To explore this concern, we generated learning curves for the groups TT10 and DM09 and compared these curves to the results for three fixed models: the default model, the best model from another group, and the model trained on all 30 games from the current group. (The error rate did not appear to decrease beyond 30 games, and so this last result is effectively the asymptote of the learning curve.) Each curve is the average of 30 runs in which we randomly selected N games for training and used the remainder for testing (except in the case of $N = 1$, in which each game was used once for training). Generating the curves in this way requires the assumption that game order is insignificant (i.e., no trend of changes as agents adapt over time), which is certainly the case with the agent binaries used and also appears to be the case in actual competition data. Up to 16 games are used for training, as until 2010 only 16 or 18 games were played per round. The results are shown in Figures 5.42 through 5.45.

Not surprisingly, when only a few games are available for training, learning online is much worse than using a fixed model. In three out of four cases, the error from learning online reached the error of the best fixed model after about 8 games, while in the fourth case (the component model for DM09), the error never sank that low. Overall, if forced to choose between using online learning or a reasonable fixed model, the fixed model appears to be a better choice. These results are similar to the results we obtained when comparing our fixed models used in competition to the results of online learning during competition (determined after the fact), which is why we continued to use fixed models.

An alternative approach would be to switch from the fixed model to the online model at some point, but this would require deciding when to switch. An even more appealing possibility, as outlined in Section 1.1 of the introduction, is the idea of somehow combining the two options, such as by combining the fixed and online models or combining their training data. This idea is explored fully in Chapters 8 and 9, where I describe learning algorithms that can indeed outperform both the fixed and online models across the full learning curve.

5.5 Summary

In this chapter, I described improvements to the TacTex agent for supply chain management based on using machine learning to predict changes in computer and component prices. These improvements were shown to have significantly

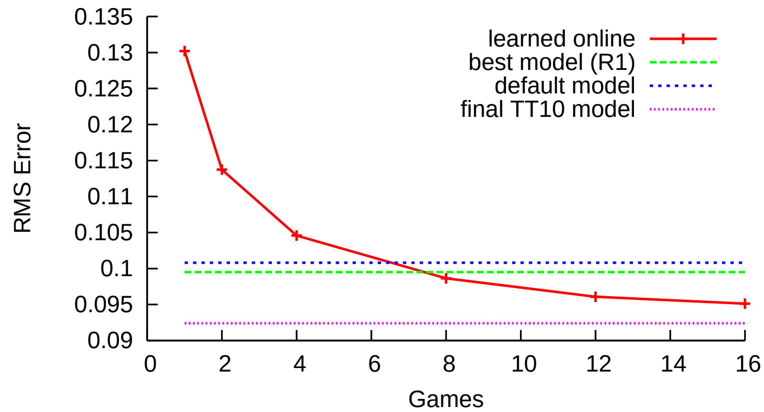

Figure 5.42: Computer price prediction error for TT10 when learned online

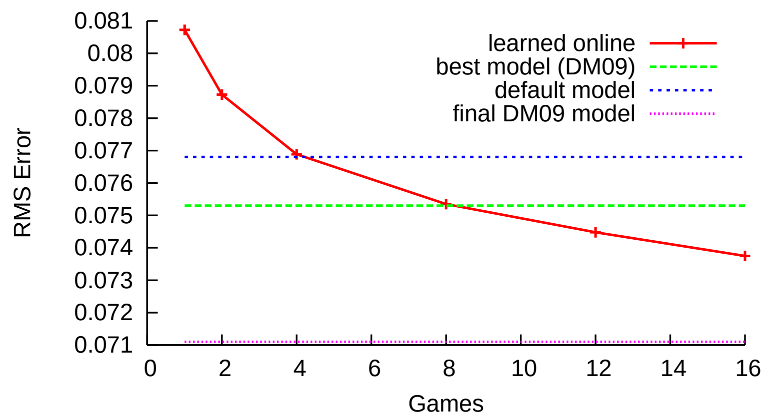

Figure 5.43: Component price prediction error for TT10 when learned online

added to TacTex's score. Next, I discussed the Prediction Challenge, which showed TacTex's prediction accuracy to be comparable to other top agents for component price predictions and clearly superior for computer price predictions. Finally, I explored the importance of the choice of training data when training models, and found that model accuracy can vary considerably depending on the similarity of the group of agents represented in the training data to the group represented by the test data. I showed that differences in model accuracy can have a large impact on agent performance, particularly for the model of computer prices. When enough games against a particular group of agents are available, training models on these

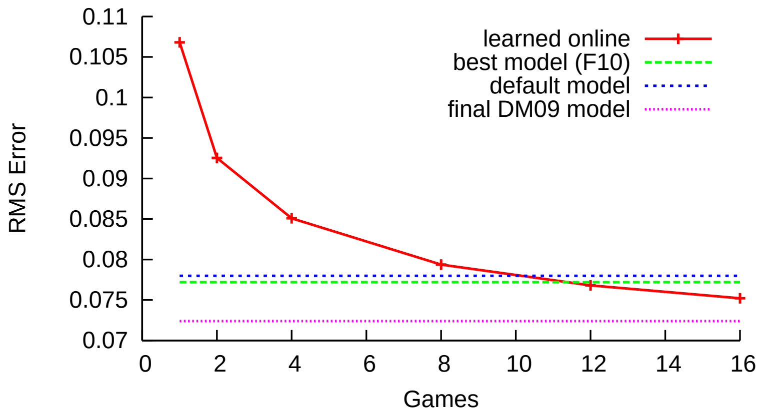

Figure 5.44: Computer price prediction error for DM09 when learned online

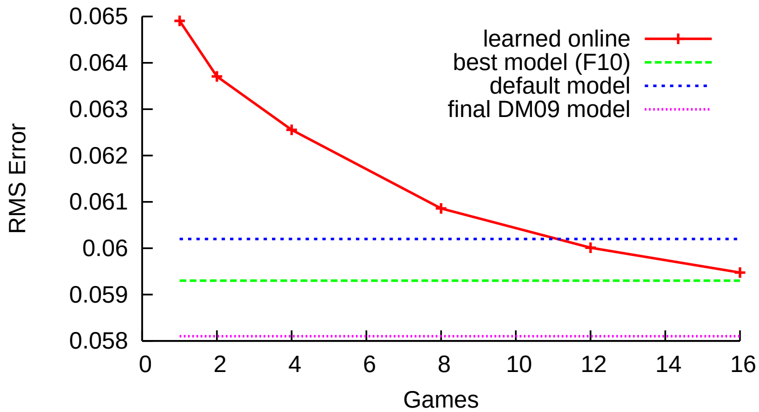

Figure 5.45: Component price prediction error for DM09 when learned online

games is clearly the best choice, but this is often not the case during the course of a TAC SCM tournament. In the next two chapters I will show that TAC AA presents a similar situation, and Chapters 8 and 9 will present a solution.

Chapter 6

The TacTex Agent for Ad Auctions

I now describe the TacTex agent for the TAC Ad Auctions scenario (TAC AA). Like the SCM agent, the TacTex agent for AA makes estimates and predictions about a variety of factors and then optimizes its actions with respect to this information. As with the TacTex agent for TAC SCM, discussion of the agent is divided into two chapters. In this chapter, I summarize the scenario, describe the full TacTex agent, and present experiments exploring the importance of each agent component. This chapter fulfills Contribution 1 for TAC AA, as outlined in Section 1.2. The following chapter then covers the use of machine learning to improve bid estimation.

6.1 TAC AA Description

Sponsored search [63] is one of the most important forms of Internet advertising available to businesses today. In sponsored search, an advertiser pays to have its advertisement displayed alongside search engine results whenever a user searches for a specific keyword or set of keywords. An advertiser can thereby target those users who might be interested in the advertiser's products. Each of the major search engines (Google and Microsoft) implements sponsored search in a slightly different way, but the overall idea is the same. For each keyword, a *keyword auction* [68] is run in which advertisers bid an amount that they are willing to pay each time their ad is clicked, and the order in which the ads are displayed is determined by the ranking of the bids (and possibly other factors).

Running a successful keyword advertising campaign can be difficult. An advertiser must choose the keywords of interest and its bids for each one based on an understanding of customer behavior, competitors' bidding patterns, and its own advertising constraints and needs, all of which can change over time. As a result, much effort has gone into developing automated strategies that can bid intelligently. A number of companies offer software for such a purpose, and the problem has also attracted a growing number of researchers. TAC AA aims to provide a realistic, competitive ad auctions environment in which independently created bidding agents can be tested against each other over the course of many simulations, just as TAC SCM does for the supply chain setting.

I now provide a summary of those parts of the TAC AA game that are most important for understanding the design of TacTex. This summary pertains to the 2009 game specification. (A few changes were made in 2010, as described in Section 6.9.) For full details, see the game specification [45].

6.1.1 Overview

In each TAC AA game, eight autonomous agents compete as advertisers to see who can make the most profit from selling a limited range of home entertainment products over 60 simulated game days, each lasting 10 seconds. Products are classified by manufacturer (*flat*, *pg*, and *lioneer*) and by component (*tv*, *dvd*, and *audio*) for a total of nine products. Search engine *users*, the potential customers, submit queries consisting of a manufacturer and a component, although either or both may be *null*, i.e. missing. There are thus 16 total query types, divided into three *focus levels*: F0 (the query with both manufacturer and component null), F1 (the six queries with one null and one specified), and F2 (the nine queries with both specified). Each day, for each of the 16 query types, a keyword auction is run. For each auction, an advertiser submits i) a (real-valued, non-negative) bid indicating the amount it is willing to pay per click, ii) an ad, and iii) a spending limit (optional). Ads can be either *targeted* (specifying both a manufacturer and product) or *generic* (specifying neither). (The set of 16 (bid, ad, spending limit) tuples can be said to be an advertiser's action space, and Section 6.7 essentially describes how TacTex maps its observations into this space each day.) The top five bidders have their ads shown in order, but if an advertiser hits its spending limit (as a result of having its ad clicked enough times), its ad is not shown for the rest of the day, and all advertisers with lower bids have their ads move up one position. Bids must exceed a small reserve price.

6.1.2 Users

There is a fixed pool of users, each of which remains interested in a specific product throughout the game and submits only queries corresponding to this product. However, users cycle through states corresponding to the focus levels according to a specified transition model. Users begin in a non-searching (NS) state, and can then transition through a searching (IS) state (which may submit a query of any focus level but will not make a purchase) to one of three buying states (F0, F1, or F2, each of which submits a query of the corresponding focus level and makes a purchase with a probability that increases with the focus), and eventually back to the non-searching state. For each product, the total number of users in any state can vary widely and rapidly.

6.1.3 Click model

Every searching or buying user submits one query per day and then proceeds through the resulting ads in order of advertiser ranking. When an advertiser's ad is shown, it is said to receive an *impression*, but not all impressions result in clicks. The default user behavior is as follows. If a user submitting query q reaches the ad of advertiser a, the probability of a click is e_q^a, a hidden parameter drawn randomly at the start of each game. If the user clicks, it will then *convert* (make a purchase) with a probability dependent on the user's focus level. For each conversion, the advertiser receives \$10. (This amount is technically the sales profit before considering advertising costs, but I will simply refer to it as the agent's *revenue*). If the user does not convert, it proceeds to the next ad with probability γ_q, another randomly drawn, hidden game parameter. Thus, the higher the position of the ad, the more likely it is to be clicked. A number of factors can modify this default behavior. First, if an advertiser's ad is targeted, the click probability is raised or lowered depending on whether the ad matches the product desired by the user. Second, each advertiser has a component and manufacturer specialty. If the product desired by the user matches the component specialty, the conversion probability is increased, and if it matches the manufacturer specialty, the advertiser's revenue is increased. Finally, the conversion probability decreases if the advertiser has exceeded its capacity, as described below.

6.1.4 Auctions

Ads are ranked using a generalized second price auction. Rather than ranking ads solely by bids, the search engine also considers click probability. If for query type q an advertiser's bid is b_q and its default click probability is e_q^a, then its *squashed bid* is defined as $(e_q^a)^\chi b_q$, where χ is a random but known game parameter. Ads are ranked by squashed bid, and each time an advertiser's ad is clicked, it pays the minimum amount it could have bid while still beating the squashed bid of the advertiser ranked below it.

6.1.5 Capacity

Each advertiser is assigned a capacity c which serves as a soft constraint on how many products it can sell (of any type) over a five day period. Whenever an advertiser's ad is clicked, if the number of products n sold over five days (including those sold so far on the current day) exceeds c, then the conversion probability is multiplied by a *capacity factor* equal to 0.995^{n-c}. Note that the capacity factor changes during the day as the advertiser sells more products.

6.1.6 Information

Advertisers must operate in the face of limited information about customers and competitors. For each query type, the advertiser receives a daily report stating how many impressions, clicks, and conversions the advertiser received and the average cost per click (CPC). The only other information available is a report on the ad used by each advertiser and the average position of that ad. An advertiser that wishes to increase its number of clicks would therefore have little information about how much it would cost to increase the position of its ad or how many clicks it might expect in the new position. Advertisers are also unaware of the types (specialties and capacities) of other advertisers.

6.1.7 Example

Table 6.1 shows the results for the query *null:dvd* from a sample game day. The eight advertisers are shown in order of their squashed bids, which differs from the order of the true bids due to differing e_q^a values. The ads and spending limits (where used) of each agent are also shown. The results of these actions are shown on the right side of the table: the cost per click, impressions, clicks, and conversions. In addition, the *impression range* column shows a graphical representation of the period for which each advertiser's ad was shown, with the day progressing from left to right. On this day, 718 users submitted the query *null:dvd*, but due to spending limits, only two agents, QuakTAC and munsey, received the full 718 impressions. TacTex was the first agent to hit its spending limit (after a single click - this was a probe, as described later). At that point, all lower advertisers increased by one position, and since epflagent reached the fifth position, its ad began to be shown. Hence, the impression range column shows epflagent starting where TacTex

	Agent actions				Results					
Advertiser	*Bid*	*Sq bid*	*Ad*	*Sp lim*	*CPC*	*Imps*	*Clks*	*Cnvs*	*Impression range*	*Avg pos*
MetroClick	0.315	0.109	generic	50.93	0.310	426	164	16		1.000
QuakTAC	0.266	0.107	pg:dvd	-	0.194	718	156	6		1.593
TacTex	0.235	0.091	generic	0.236	0.201	77	1	0		3.000
UMTac09	0.216	0.078	generic	7.583	0.209	700	36	6		2.719
munsey	0.190	0.075	generic	-	0.174	718	16	2		3.675
epflagent	0.214	0.068	generic	-	0.184	641	3	0		4.510
AstonTAC	0.158	0.059	generic	500.0	0.133	292	1	0		4.938
Schlemazl	0.062	0.020	flat:dvd	5.617	-	0	0	0		-

Table 6.1: Results for the query *null:dvd* from one game day of the 2009 TAC AA finals

stopped. Although Schlemazl reaches the fifth position at the end of the day, its ad is not shown because its bid is below the reserve. Finally, the average position for each advertiser is shown. Note that the average positions are not in the same order as the squashed bids, and that the average is only for the period in which the ad was shown (thus never above 5). From this table, the only information available to TacTex was its own row (except for the squashed bid) and the ad and average position columns. Much of TacTex's computational effort is devoted to estimating the rest of this information to facilitate the decision-making process.

6.2 TacTex Overview[1]

TacTex was the winner of the 2009 and 2010 TAC AA competitions. As the agent changed little between years, the the agent description that follows is accurate for both versions of the agent.

At a high level, TacTex operates by making predictions or estimates concerning various factors (such as unknown game parameters, user populations, and competitor bids) and then by finding the optimal actions given this information.

[1] I would like to thank Doran Chakraborty for his contributions to TacTex. In particular, Doran assisted in the development of the User Model (6.4) and designed the ad prediction method (6.5.2) and first bid estimator (6.5.3.1) contained in the Advertiser Model.

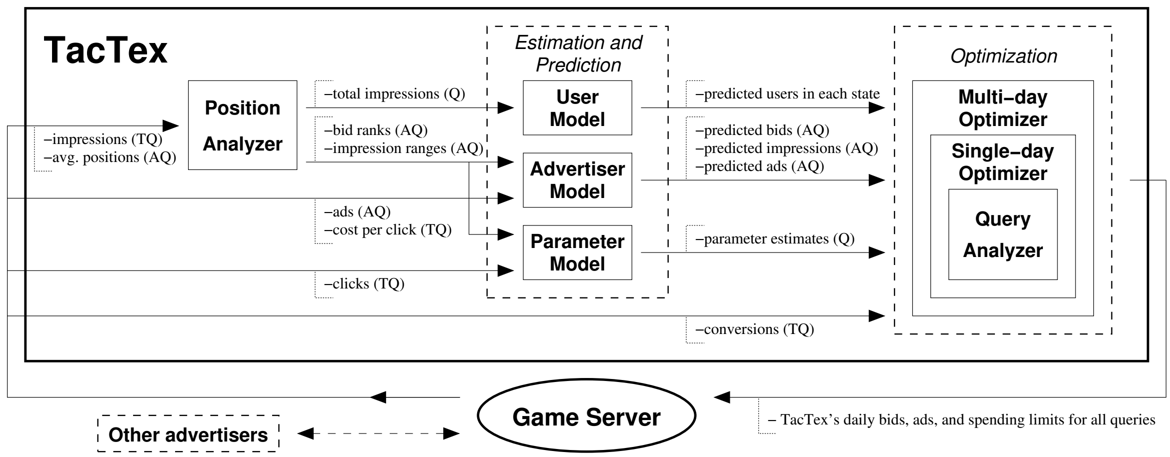

Figure 6.1: Flow of information in TacTex. T = TacTex only, A = all advertisers, Q = all queries.

These tasks are divided among a number of modules that I describe in detail in the following sections. Here, I give an overview of these modules. Figure 6.1 depicts the relationship between the modules, including the inputs and outputs of each.

At the start of each new day, the game server sends TacTex a report on the results of the previous day. The first module to be called is the Position Analyzer, a preprocessor that converts some of this information into a more useful format. The goal of the Position Analyzer is essentially to reconstruct the impression range column of Table 6.1 for each query type.

TacTex then performs all necessary prediction and estimation using three modules. The User Model uses the total number of queries for each query type to estimate the composition of each of the nine user populations. From these estimates, predictions about future user populations can be made. The Advertiser Model takes information relating to the actions of other advertisers and predicts the actions these advertisers will take in the future. The Parameter Model maintains estimates of unknown game parameters by finding those parameters that best fit the known auction outcomes.

Finally, TacTex must use these predictions and estimates to choose the optimal bids, ads, and spending limits to submit to the game server for the next day. The Query Analyzer uses the information received to compute the expected outcomes of actions, such as how many clicks and conversions would occur for a given query type. If there were no capacity factor, then TacTex could optimize for each query type independently, but instead it must choose how to allocate its available capacity among query types and even among multiple days. The Multi-day Optimizer is responsible for dividing capacity among the remainder of the game days, and it calls the Single-day Optimizer to divide each day's capacity among query types using the information provided by the Query Analyzer.

6.3 Position Analyzer

For each query type, the Position Analyzer takes the average advertiser positions and attempts to extract i) the ranking of the squashed bids, and ii) the first and last impression for each advertiser. Each advertiser's average position can be expressed as a function of the bid rankings and first and last impressions of all other advertisers, and while this system of equations cannot be solved directly, it is possible to efficiently search the space of unknown values to find a correct or nearly correct solution. The details of the search algorithm are given in the Appendix.

6.4 User Model

The User Model maintains estimates of the user population states by using a particle filter (specifically a Sampling Importance Resampling filter [1]) for each of the nine populations. (See Section 5.1.1 for a brief explanation of particle filtering.) In this case, each of the 1000 particles used per filter represents a distribution of the $10,000$ users of that type among the six individual user states (NS, IS, F0, F1, F2, and T). At the beginning of the game, the particles are chosen to reflect the possible populations resulting from the initialization process performed by the game server. Each succeeding day, a new set of particles is generated from the old. For each new particle to be generated, an old particle is selected at random based on weight, and the new particle's user distribution is randomly generated from the old particle based on the user transition dynamics. These dynamics are known with the exception of the probability of a user transitioning to the Transacted state as a result of a conversion. This probability depends on the ads seen by the user, and thus on the behavior of the advertisers, but we can estimate it fairly accurately from past games.

The new particles are then re-weighted based on TacTex's observations. The daily observations that depend on the user populations are the total impressions, clicks, and conversions for each of the 16 queries. Some of these observations are more informative and straightforward to use than others. Observations for queries in which the manufacturer or component are not specified (F0 and F1 queries) are less informative because they represent the behavior of users from multiple populations. Conversion rates depend on the position and capacity factor at the time of the click, and suffer from small sample sizes. The most informative observations are the total impressions for queries that specify both manufacturer and component (F2 queries), and we found that we were able to estimate user populations accurately using only these observations. Note that unless TacTex had its ad displayed for every impression for a query type (an uncommon case), the total impressions for that query type must be determined using the Position Analyzer.

The number of impressions for an F2 query is the number of users submitting that query, which is the number of F2 users plus those IS users that chose the F2 query. An IS user has a $1/3$ probability of choosing the F2 query, so the probability of observing N total impressions when there are x_i IS users and x_{f2} F2 users is the probability that $N - x_{f2}$ of the x_i IS users choose the F2 query, which can be determined from the binomial distribution $B(x_i, 1/3)$ or (in our case) estimated using the normal approximation to this binomial distribution, $N(x_i/3, 2x_i/9)$. Each particle has its weight set to this probability, and weights are then normalized to sum to one.

The resulting set of particles represents our estimated probability distribution over the user population state on the previous day. To obtain the expected user population n days in the future, we update each particle $n+1$ times according to the user transition dynamics and take the weighted average of the particles.

Figures 6.2 and 6.3 show the estimated and actual number of users in the IS, F0, F1, and F2 states for two different products in one game from the 2009 finals. The game server models users as exhibiting occasional bursts of interest in a product (i.e., moving from the NS to IS state in large numbers), and these bursts can be seen in the IS user plots. If these bursts can be detected, then the estimated number of IS users will be very accurate, and the estimates of users in other states will follow. In Figure 6.2, these bursts are detected perfectly, and accuracy for all four user states is extremely good. For the product depicted in Figure 6.3, TacTex's estimates of total daily impressions were poor on some days, and on these days the User Model could not be sure whether a burst occurred. As a result, the IS estimate shows a number of small "blips", where some particles reflected a burst and others did not, and one true burst was missed. Nevertheless, the User Model was able to recover on succeeding days, and the estimates for all four states remain good overall.

6.5 Advertiser Model

The Advertiser Model makes three types of predictions about the actions of the competing advertisers.

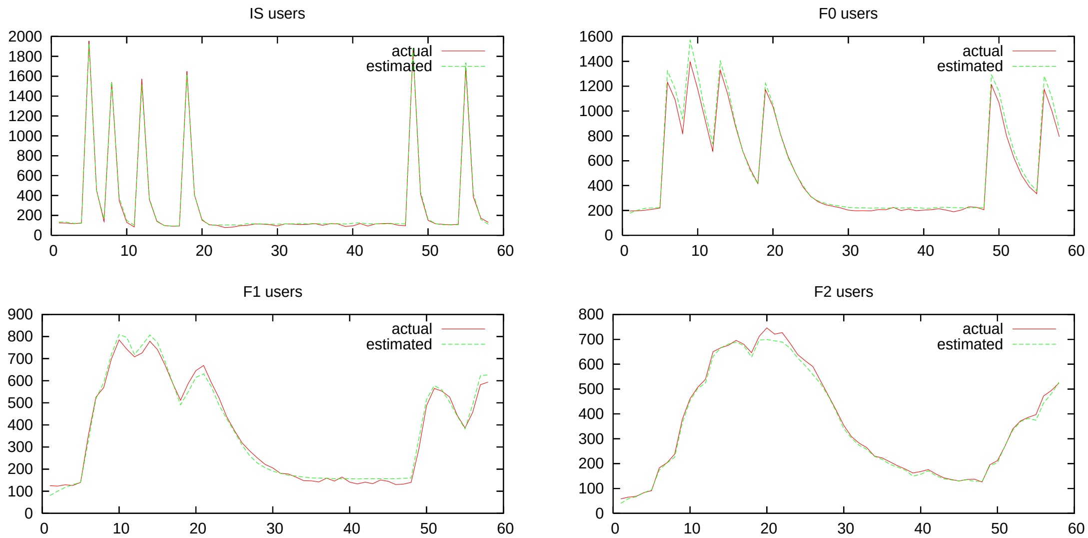

Figure 6.2: User population estimates for one product

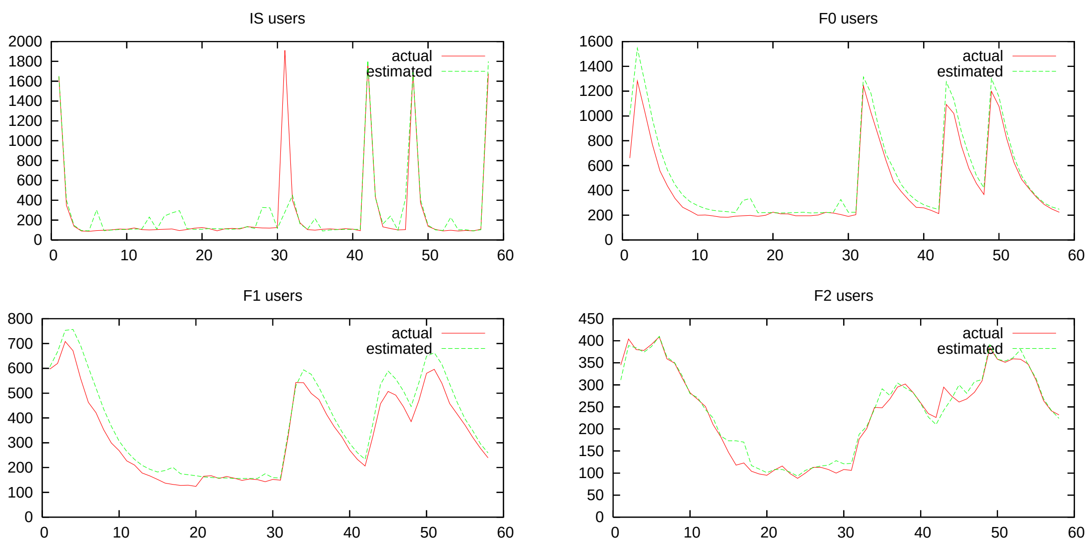

Figure 6.3: User population estimates for a second product

6.5.1 Impression Predictions

In addition to predicting the bids of other advertisers, it is also important to predict whether and when they will hit their spending limits. If other advertisers set low spending limits, a relatively low bid could still result in a high average position and number of clicks. For each query type, the Advertiser Model predicts the maximum number of impressions that each advertiser could receive before hitting its spending limit. In general, the Advertiser Model predicts that this maximum will be the same number of impressions as the advertiser received on the previous day; however, in cases in which an advertiser did not hit its spending limit by the day's final impression, we assume that the advertiser effectively had no spending limit and will also not hit its spending limit on the coming day. Information about the impressions received by other advertisers is provided by the Position Analyzer.

6.5.2 Ad predictions

The Advertiser Model also predicts the ads (targeted or generic) that other advertisers will choose. For each query type, the Advertiser Model maintains a count for all the ads it has seen so far from each advertiser. The predicted ad for that query is then the majority ad, i.e., the ad having the highest count amongst all posted ads for that query type.

6.5.3 Advertiser Bid Estimation

The third task performed by the Advertiser Model is to maintain estimates of the bids submitted by each advertiser for each query and then to predict what future bids will be. Advertiser bid estimation is a hard problem because the bidding dynamics of other advertisers are unknown–while bids often change only gradually, it is not uncommon for large jumps to occur. In addition, the Advertiser Model receives only partial information about the bids of other advertisers (the bid ranks and TacTex's CPC). During the development of TacTex, we created two very different and independently designed bid estimators. Our preliminary testing did not show either approach to be superior; however, we found that an ensemble approach that averaged the output of the two estimators outperformed either one alone, and so the Advertiser Model uses both estimators in this fashion. The primary difference between the two is that the first estimator models all advertisers' bids jointly, while the second models bids independently.

The estimates obtained are for the bids active on the previous game day; however, our goal is to predict future bids for use in planning. Currently, the Advertiser Model simply assumes that the most recently estimated bids will persist for the rest of the game, but improving these predictions is an important area for future development.

6.5.3.1 First bid estimator[2]

The first approach (Algorithm 1) uses particle filtering to estimate the bids of other advertisers. We use one particle filter for each of the 16 query types. Each of the 1000 particles per filter represents one set of bids for all other advertisers for that query type. Associated with the particles is a probability distribution that gives the likelihood of each particle representing the current state. On each new day, each particle filter samples from the underlying distribution to obtain the next set of particles. Then it updates each particle based on the observations received that day, i.e., the cost per click and bid rankings. Once the particles have been updated, the filter recomputes the probability distribution for the new set of particles.

The sampling step is straightforward. The next step in particle filtering is

[2] Doran Chakraborty designed the first bid estimator and wrote the initial draft of the material in this section.

to update each particle using the known dynamics. In the absence of such known dynamics, we propose a departure from the traditional vanilla particle filter and use a part of the observation to do the update. Let cpc_{t+1} and r_{t+1} be the cost per click and ranking for that query seen on day $t + 1$. We use these values to reset some of the bids made by other advertisers in some particles (where necessary) in an attempt to improve the respective particle. On each iteration of the update, the bid, b^x_{t+1}, of an advertiser x is adjusted while holding the bids of the other advertisers fixed. The bids are adjusted for only those cases where the order of a bid is incorrect with respect to the known bid ranking. Algorithm 2 gives an outline of how the bid adjustment is done. The two cases where the order is correct and there is no need for bid adjustment are:

$$
\begin{aligned}
r^x_{t+1} > (r^{TacTex}_{t+1} + 1) \quad \wedge \quad b^x_{t+1} < cpc_{t+1} \\
r^x_{t+1} < r^{TacTex}_{t+1} \quad \wedge \quad b^x_{t+1} > b^{TacTex}_{t+1}
\end{aligned}
\tag{6.1}
$$

The conditions when b^x_{t+1} needs to be updated, and how these updates are made, are mentioned below. $rand(a, b)$ denotes a random draw from the range (a, b). z denotes the particle and $z(r)$ denotes the bid of the advertiser ranked r in z.

$$
b^x_{t+1} = \begin{cases} cpc_{t+1} & \text{if } r^x_{t+1} = r^{TacTex}_{t+1}{+}1 \\ rand(0, \text{least bid value in } z) & \text{if } r^x_{t+1} = \text{undefined} \\ rand(z(r^x_{t+1}{+}1), z(r^x_{t+1}\text{-}1)) & \text{otherwise} \end{cases}
$$

Note that the "otherwise" case excludes the conditions mentioned in Equation 6.1. The whole process is repeated a fixed number of times, holding one advertiser fixed each time, with each iteration improving upon the former (20 iterations is sufficient). At the end of this update step, we have a better particle having closer predictions of other advertiser bids.

Next comes the step of recomputing the probability distribution of the sampled particles (Algorithm 3). Although the true likelihood of a particle whose ranking does not match the true ranking r_t is zero, there may be few particles with

Algorithm 1: ADVERTISER-BID-ESTIMATOR

begin
  input : $\vec{z}, Pr(\cdot), cpc, ranks$
  for $\forall z \in \vec{z}$ **do**
    $z \leftarrow$ ADJUST-BIDS$(z, cpc, ranks)$
  Sample n particles from the filter
  for $\forall z \in \vec{z}$ **do**
    $Pr(z) \leftarrow$ RECOMPUTE-DISTRIBUTION$(z, ranks)$
  normalize $Pr(\cdot)$

the correct ranking, and so we instead use a likelihood function designed to give some weight to all particles. We compute the difference of ranking for each advertiser from the two available sources, i.e., r_t and the rank from z. For a distance δ, we define $\kappa(\delta) = exp(-\frac{\delta^2}{4.9})$. The likelihood of each particle is set to the product of these $\kappa(\delta)$ values over all advertisers, and thus the particles whose predicted rankings are closer to r_t get assigned higher values. These values are normalized over all 1000 particles to give the true probability distribution captured by the particles.

6.5.3.2 Second Bid Estimator

The second bid estimator differs from the first in two main ways. First, for each query it maintains separate bid distributions for each advertiser, rather than a single joint distribution over all bids. Second, while both estimators approximate continuous distributions using discrete distributions, the second filter does so using a fixed set of bids rather than a changing set of particles. The bid space $[0, 3.75]$ is discretized into values v_1 through v_{100} by setting $v_i = 2^{i/25-2} - 0.25$ (thus $v_{50} = 0.75$). Discretizing the bid space in this way allows better coverage of low bids, which are most common, while still maintaining the ability to represent very high bids, and it also simplifies our modeling of changes in bids, described below.

On day $t + 1$, for each advertiser x, we wish to estimate the distribution of

Algorithm 2: ADJUST-BIDS

begin
    input : $z, cpc, ranks$
    output: z
    LO-BID $\leftarrow$ set a low bid limit
    $size \leftarrow$ no of advertisers in $ranks$
    $r^{TacTex} \leftarrow$ own rank from $ranks$
    $b^{TacTex} \leftarrow$ own bid for q
    $count \leftarrow$ no of iterations
    while *count times* **do**
        for *($\forall x \in$ advertiser set)* **do**
            $r^x \leftarrow$ rank of x from $ranks$
            switch do
                case $r^x = r^{TacTex} + 1$
                    $b^x \leftarrow cpc$
                case $r^x = -1$ *(undefined)*
                    $b^x \leftarrow rand($LO-BID, $z(size))$
                case $r^x > (r^{TacTex} + 1) \wedge b^x \geq cpc$
                    $b^x \leftarrow rand(z(r^x + 1), z(r^x - 1))$
                case $r^x < r^{TacTex} \wedge b^x \leq b^{TacTex}$
                    $b^x \leftarrow rand(z(r^x + 1), z(r^x - 1))$

Algorithm 3: RECOMPUTE-DISTRIBUTION

begin
    input : $z, ranks$
    output: $Pr(\cdot)$
    $Pr(z) \leftarrow 1$
    for $\forall a \in$ *advertiser set* **do**
        $\delta \leftarrow$ difference of a's rank between z and $ranks$
        $\delta_f \leftarrow \frac{diff}{7}$
        $Pr(z) \leftarrow Pr(z) \times exp(-\frac{\delta_f^2}{C})$
    normalize $Pr(\cdot)$

the new bid, b^x_{t+1}, over these discrete v values, conditional on the observed ranking r_{t+1}, previous bids b^x_1 ... b^x_t, and the bids of other advertisers, B^{-x}_{t+1}. We make the simplifying assumptions that r_{t+1} and b^x_1 ... b^x_t are conditionally independent given b^x_{t+1}, and that B^{-x}_{t+1} and b^x_1 ... b^x_{t+1} are independent. Applying Bayes' rule twice and rearranging, we derive:

$$Pr(b^x_{t+1} = v_i | r_{t+1}, B^{-x}_{t+1}, b^x_1 ... b^x_t) \propto$$
$$Pr(r_{t+1} | B^{-x}_{t+1}, b^x_{t+1} = v_i) Pr(b^x_{t+1} = v_i | b^x_1 ... b^x_t) \tag{6.2}$$

The first term in the R.H.S. of Equation 6.2 is the probability of the observation while the second term is the transition model of bids for x, both unknown.

We model bid transitions by assuming that bids change in one of three ways. First, with 0.1 probability, b^x_{t+1} jumps uniformly randomly to one of the v_i values. This case covers sudden jumps that are difficult to model. Next, with 0.5 probability, b^x_{t+1} changes only slightly from b^x_t. We assume that the probability of changing from v_i to v_j is proportional to $\phi_{0,6}(|i - j|)$, where $\phi_{0,6}$ is the density function of the zero-mean normal distribution with variance 6. Because we discretized the bid space in such a way that bids increase exponentially, the use of this distribution reflects the assumption that the logarithms of the ratios of successive bids are distributed normally with zero mean. Finally, we assume that with 0.4 probability, the bid changes according to a similar distribution, but the change is with respect to the bid 5 days ago, b^x_{t-4}. This case captures the fact that bids often follow 5 day cycles due to the 5 day capacity window. The probabilities for these three cases were chosen to provide robustness to a variety of agent behaviors in pre-competition experiments. Let $tr(j, i)$ denote the the resulting probability of the bid transitioning to v_i from v_j using the above normal distribution and normalizing.

Summing the three cases gives us the following:

$$P(b^x_{t+1} = v_i | b^x_t = v_j, b^x_{t-4} = v_k) \ = \tag{6.3}$$

$$0.1 \cdot 0.01 + 0.5\, tr(j,i) + 0.4\, tr(k,i)$$

and so we can find the probability of each bid by summing over our previous distribution estimates:

$$P(b^x_{t+1} = v_i) = 0.1 \cdot 0.01 + 0.5 \sum_{j=1}^{100} P(b^x_t = v_j) tr(j,i) + \tag{6.4}$$
$$0.4 \sum_{k=1}^{100} P(b^x_{t-4} = v_k) tr(k,i)$$

Our estimate for b^x_1 is initialized to a distribution consistent with observed game data, and when $t < 5$, we substitute b^x_1 for b^x_{t-4}.

The observation probabilities are now conditioned on a single advertiser's bid, rather than a set of bids as in the first bid filter. Let y be another advertiser in the game apart from x. If advertiser y is TacTex, then we know the bid; otherwise we have a distribution representing our estimate of the bid for y. Thus the conditional probability of the set of rankings r_{t+1} given a fixed $b^x_{t+1} = v_i$ and a fixed value of the distribution B^{-x}_{t+1} is :

$$P = \Pi_{\forall y \neq x} \begin{cases} Pr(b^y_{t+1} > v_i) & \text{if } r^x_{t+1} > r^y_{t+1}, \\ Pr(b^y_{t+1} < v_i) & \text{if } r^x_{t+1} < r^y_{t+1}, \\ 1 & \text{otherwise} \end{cases} \tag{6.5}$$

where P denotes $Pr(r_{t+1} | b^x_{t+1} = v_i, B^{-x}_{t+1})$. Note that ranks will only be equal if neither advertiser had any impressions; in this case we have no information about the relative bids. Also, whenever y is TacTex, the R.H.S. will be 1 or 0 since we know our own bid. In addition, in some cases it is possible to use TacTex's cost per click to exactly determine the bid of the advertiser below it. Finally, we have been treating B^{-x}_{t+1} as if it were known, but in fact these are the other advertisers' bids that we are trying to estimate simultaneously. We address this problem by applying Equation 6.2 for 10 iterations, using the latest estimates for each bid distribution, as this resulted in sufficient convergence in testing. Algorithm 4 summarizes the update procedure used each day by the second bid estimator.

Algorithm 4: UPDATE-DISTRIBUTIONS

begin
- **input** : rankings r_{t+1}, TacTex's bid and cpc
- **for** *each advertiser* x **do**
  - /* let T_i^x be $Pr(b_{t+1}^x = v_i | b_1^x ... b_t^x)$ */
  - **for** $i = 1 \ldots 100$ **do**
    - set T_i^x according to Equation 6.4
- **for** $n = 1 \ldots 10$ **do**
  - **for** *each advertiser* x **do**
    - **for** $i = 1 \ldots 100$ **do**
      - /* let O_i^x be $Pr(r_{t+1} | B_{t+1}^{-x}, b_{t+1}^x = v_i)$ */
      - **if** $n = 1$ **then**
        - $O_i^x = 1$
      - **else**
        - set O_i^x according to Equation 6.5
  - **for** *each advertiser* x **do**
    - **for** $i = 1 \ldots 100$ **do**
      - $Pr(b_{t+1}^x = v_i) \leftarrow O_i^x T_i^x$
    - normalize $Pr(\cdot)$

6.6 Parameter Model

Recall that for each query type q, the parameter γ_q represents the probability that a user will progress from one ad to the next, while each advertiser a has a parameter e_q^a that affects the probability of a user clicking its ad (and thus also its squashed bid). Given the bid rankings and impression ranges computed by the Position Analyzer and the User Model's population estimate, we can determine the distribution over the number of clicks that TacTex would receive for any set of these parameter values. The Parameter Model estimates the values of γ_q and e_q^{TacTex}, as these are the parameters that have the most impact on TacTex and about which our observations provide the most information. There is insufficient information to effectively estimate e_q^a values of other advertisers, and so we assume they equal the mean possible value. To perform estimation, the Parameter Model maintains a joint distribution over (γ_q, e_q^{TacTex}) pairs by discretizing the possible space of values uniformly and setting the likelihood of each pair to be proportional to the probability of all observations given that pair, that is, the product of the probabilities of each day's number of clicks. When performing calculations involving these parameters, the Parameter Model takes the weighted average for each parameter as its estimate. While these estimates do not generally converge to the correct values by the end of a game (and might not ever be expected to given our decision to not estimate other advertisers' e_q^a values), they are much more accurate on average than simply assuming the expected parameter values–typically the estimate falls somewhere between the true parameter value and the expected value.

6.7 Optimization

To this point, we have described those modules that estimate the game state and make predictions about the future. We now turn to the challenge of using this information to select actions. In particular, each day TacTex must choose bids, ads, and spending limits for each query. The key factor in the optimization process is the capacity factor. Recall that while there is no hard cap on capacity, exceeding a certain number of conversions results in a reduced conversion rate. Beyond some point, net marginal returns per conversion can become negative. As a result, TacTex performs optimization by reasoning about conversions and then choosing actions expected to result in those conversions, rather than reasoning directly in the space of possible actions.

The optimization process consists of three levels: a Multi-day Optimizer (MDO), a Single-day Optimizer (SDO), and a Query Analyzer (QA). Because we want to maximize profit for the entire game, not a single game day, the top-level decision that must be made is how many conversions to target on each remaining game day, and this decision is made by the MDO. Computing the expected profit for a given day and conversion target requires deciding how to divide the conversions among the 16 query types, and the MDO calls the SDO to perform this task. Finally, the SDO calls the QA to i) determine the bid, ad, and spending limit that are expected to result in a given number of conversions, and ii) compute the expected cost and revenue from those conversions. We describe these three levels from the bottom up.

6.7.1 Query Analyzer

For any given bid, ad, and spending limit, it is fairly straightforward to determine the expected cost, revenue, and conversions from a specific query type. The QA does this by taking the expected user population, iterating through all impressions, computing our position and CPC, and then computing the probability that the user i) reaches our ad, ii) clicks on it, and iii) converts. (Note that at this level we are not considering the capacity factor, which may lower the conversion rate.) However, the problem we face is essentially the reverse: the QA is given a conversion target and needs to determine the bid, ad, and spending limit that will produce those conversions in the most profitable way. Up to a certain point, raising either the bid or the spending limit will increase the number of conversions, while the effect of ad choice depends on the user population, so there may be a number of ways to reach a given number of conversions.

We simplify matters by using no spending limits. During the course of a day, the expected profit per impression can only increase as other agents hit their spending limits. If an agent above us hits its spending limit, our position improves along with our conversion rate. (Higher positions have higher conversion rates due to a higher ratio of F users to IS users–IS users never convert and thus are more likely to reach ads at lower positions.) If the agent below us hits its limit, then our CPC is reduced, as we are participating in a generalized second price auction. It is therefore preferable to control conversions using the bid rather than the spending limit.[3]

For any given bid, we can evaluate each of the relevant ads and pick the one that gives the highest profit per conversion. As a result, the QA's primary task is to determine the bid that will result in the desired number of conversions. There is one difficulty remaining, however: because our prediction for each advertiser's bid is a point estimate, any bid between the nth and $(n+1)$st predicted bids will result in the same position, $n+1$, and the function mapping bids to conversions will be a step function. In reality, there is uncertainty about the bids of other advertisers, and we would expect this function to be continuous and monotonically increasing. We create such a function by linearly interpolating between the expected results for each position. In particular, we assume that the number of conversions expected for the nth position will result from bidding the average of the $(n-1)$st and nth bid. For $n = 1$, we use a bid 10% above the predicted highest bid, and for $n = 8$, we use a bid 10% below the predicted lowest bid. We generate functions for cost and revenue in the same way.

The complete procedure followed by the QA for each query type q is therefore as follows. First, we find the eight bids (along with corresponding optimal ads) that correspond to the eight possible positions, and determine the expected conversions, cost, and revenue for each. Next, we use linear interpolation to create functions mapping bids to conversions, cost, and revenue. Finally, for a conversion target c, we can find the bid resulting in the target from the conversions function and determine the resulting cost ($cost_q(c)$) and revenue ($revenue_q(c)$) from the corresponding functions. The ad to use is the ad corresponding to the closest of the

[3] During the 2009 and 2010 TAC AA competitions, TacTex used high spending limits as a precaution. Our experiments have shown that this was unnecessary, at least for the 2009 specification and agents, and so we omit them from the agent described here and in all experiments described below.

eight bids.

6.7.2 Single-day Optimizer

Using this information about each query type, the SDO can now determine the optimal number of conversions to target for each query type given a total daily conversion target c and the initial capacity used u. The initial capacity used (the sum from the past four days) is important because it, along with the total conversion target, determines the capacity factor, which can in turn have a large impact on the profit from each conversion and the optimal solution. To illustrate, suppose we need to choose between targeting a single conversion from query type A with an expected revenue of 10 and expected cost of 8.5, and a single conversion from query type B with an expected revenue of 15 and expected cost of 13.3. With a capacity factor of 1 for this conversion, the profits would be 1.5 and 1.7, respectively. With a capacity factor of 0.9, however, the profits would be 0.5 and 0.2, changing the optimal choice from B to A. It is also important to note that although we are targeting one conversion, with a capacity factor of 0.9 we would actually only expect 0.9 conversions; this issue we will be addressed further below.

Computing the precise impact of the capacity factor is difficult because it decreases after each conversion, meaning that we would need to know when a conversion occurred to compute its profit. We solve this problem by making the simplifying assumption that the day's average capacity factor applies to each conversion. We denote this value $\bar{d}(u, c)$ because it can be computed from the initial capacity used and the total conversion target; in fact, we precompute all possible $\bar{d}(u, c)$ values before the game begins. The goal of the SDO is thus to find values of c_q maximizing $\sum_q [\bar{d}(u, c) revenue_q(c_q) - cost_q(c_q)]$, where the c_q values correspond to the query types and sum to c. Again, although we are targeting c conversions, we would actually only expect $\bar{d}(u, c)c$ conversions. In general we reason in terms of conversions *before* adjusting for the capacity factor, and so for clarity the term *adjusted* conversions will be used when referring to the actual number of resulting conversions. Note that u is expressed in terms of adjusted conversions, while c is not.

We are now left with a fairly straightforward optimization problem: allocating the total conversion target among the queries so as to maximize profit. This problem can be solved optimally using dynamic programming by casting it as a multiple choice knapsack problem, with each (*query type, conversion*) pair representing a single item and each query type representing a class from which only one item can be chosen. This solution is too slow for our needs, unfortunately, and so instead the SDO uses a nearly-optimal greedy solution in which we repeatedly add conversions from the most profitable query type. If it were always the case for each query type that as the number of conversions increased, the marginal profit per query type decreased, then we could add the most profitable conversion at each step and be guaranteed the optimal solution. In order to receive more conversions we must increase our bid, and thus our cost per click increases. However, as our position improves, is it possible for our conversion rate to improve enough to offset this cost, and so the marginal profit per conversion may actually increase. As a result, instead of adding a single conversion at each step of our greedy approach, we consider adding multiple conversions. For each query type, we determine the number of additional conversions (bounded above by the number of conversions remaining before we hit our target) that maximizes the average profit per additional conversion, and we then add the conversions from the query type with the highest average profit. This greedy approach is not guaranteed to be optimal, but tests show that the resulting expected profit differs from the results of the optimal dynamic programming approach by less than 0.1% on average.[4] The greedy optimizer is summarized in Algorithm 5.

[4] An extension of this greedy approach that is optimal exists and was used by the agent Schlemazl [14].

Algorithm 5: SINGLE-DAY-OPTIMIZER

begin

  input : capacity used u, capacity target c

  output: profit

  $cSum \leftarrow 0$

  for *each query type* q **do**

    obtain $cost_q()$ and $revenue_q()$ from the QA

    $c_q \leftarrow 0$

  while $cSum < c$ **do**

    Find q and $c_q^{'}$ maximizing $\frac{\bar{d}(u,c)(revenue_q(c_q^{'})-revenue_q(c_q))-(cost_q(c_q^{'})-cost_q(c_q))}{c_q^{'}-c_q}$ such that $c_q^{'} > c_q$ and $cSum + c_q^{'} - c_q \leq c$

    $cSum \leftarrow cSum + c_q^{'} - c_q$

    $c_q \leftarrow c_q^{'}$

  $profit \leftarrow \sum_q \bar{d}(u,c)revenue_q(c_q) - cost_q(c_q)$

6.7.3 Multi-day Optimizer

The SDO determines bids for any given conversion target and amount of capacity already used. Determining the bids to submit for the next day therefore requires only that we choose the conversion target. Because the bids submitted today affect not only tomorrow's profit but also the capacity remaining on future days, we cannot simply choose the conversion target myopically. To maximize expected profit over the remainder of the game, we must consider not only tomorrow's conversion target, but also the actions we will take on all succeeding days.

The MDO operates by finding the optimal set of conversion targets for the remainder of the game. The expected profit from any set of conversion targets can be determined by successively applying the SDO to each remaining game day. The goal of the MDO on day d is is therefore to find the conversion targets c_t maximizing $\sum_{t=d+1}^{59} SDO_t(c_t, u_t)$, where SDO_t returns the expected profit from applying the SDO on day t, and u_t represents the total adjusted conversions over four days preceding t (which can be computed from the c_t values). Note that planning for the entire game requires calling the QA (and thus predicting the bids of other agents) for all remaining game days, not only the next day. Currently, TacTex simply assumes that the bids predicted for the next day persist for the rest of the game, but improving these predictions is an important area for future work.

Once again, an optimal solution can be found using dynamic programming, but we choose another approach due to time constraints. The dynamic programming approach requires working backward from the last day and finding the optimal conversion target given the number of (adjusted) conversions on each of the previous four days. The MDO instead uses a form of hill climbing search to solve this optimization problem. We begin by setting each c_t value to be one-fifth of TacTex's capacity. Then for all t, we consider increasing or decreasing c_t by one and compute the expected profit in each case. We then choose the most profitable deviation over all t. This process repeats until no deviation is profitable. In comparing these two optimization approaches, we found it necessary to use a somewhat coarse degree of granularity (increments of five conversions) when implementing the dynamic programming approach due to memory limitations, and as a result the hill-climbing approach was actually slightly better than the dynamic programming approach. The optimization procedure followed by the MDO is summarized in Algorithms 6 and 7.

Once the optimal set of conversion targets is found, the MDO takes tomorrow's conversion target and submits the bids determined by the SDO. Essentially, we plan for the rest of the game and take the first step of this plan. On the next day, we repeat this process using updated information.

There is one remaining special case. It will often be the case that we are not interested in bidding on a particular query. When this happens, TacTex submits a probe bid designed to provide information about the bids of other advertisers.

Algorithm 6: Find-Profit

```
begin
    input  : day d, conversion targets c_d ... c_end, capacities used
             u_{d-1},u_{d-2},u_{d-3},u_{d-4}
    output: profit
    profit ← 0
    for t = d ... end do
        u ← u_{t-1} + u_{t-2} + u_{t-3} + u_{t-4}
        profit ← profit + Single-Day-Optimizer(u, c_t)
        u_t ← d̄(u, c_t)
```

The bid chosen is one that we expect to be the nth ranked bid, where n is the rank between 2 and 6 that we have hit least recently. We set a spending limit equal to the bid so that we will likely only receive a single click.

6.7.4 First Two Days

On the first two game days, we have not yet received any information about auction results, and so we cannot use the bidding strategy described above. Many agents use hard-coded bids, but we choose our bids using information from past games. For each query type, we choose an average position to target using a simple heuristic: the target begins at 5, and if the query type matches one of our specializations, the target decreases by an amount depending on our capacity. We compute the bid that we expect to result in this position by performing linear regression on data observed in previous games to learn a function mapping bids to positions. A separate function is learned for each day and focus level.[5]

[5] In all experiments described below, fixed bids were used on the first two days.

Algorithm 7: MULTI-DAY-OPTIMIZER

begin
  input : day d, capacities used u_{d-1},u_{d-2},u_{d-3},u_{d-4}
  output: conversion target c_d
  for $t = d \ldots end$ **do**
    $c_t \leftarrow \frac{total\ capacity}{5}$
  while *maximum profit improves* **do**
    for $t = d \ldots end$*,* $change \in \{1, -1\}$ **do**
      $c_t \leftarrow c_t + change$
      $profit_t^{change} \leftarrow$ FIND-PROFIT$(d, c_d \ldots\ c_{end}, u_{d-1} \ldots\ u_{d-4})$
      $c_t \leftarrow c_t - change$
    find t and $change$ maximizing $profit_t^{change}$
    $c_t \leftarrow c_t + change$

6.8 2009 Agent Performance

Having completed the description of TacTex, I now analyze the performance of the 2009 agent. First, I present competition results. I then give the results of a number of controlled experiments designed to measure the importance of the different components of TacTex.

6.8.1 2009 Competition

The first annual TAC AA competition was held over two days at IJCAI 2009. All 15 qualifying agents participated in a round-robin style semifinal round, and then the top eight agents advanced to a final round where each agent participated in all 80 games.

TacTex finished first in both rounds. The scores (i.e., average ending balances) from the final round are shown in the table in Figure 6.4. A Wilcoxon two-sample test shows that the difference in score between TacTex and each other agent is statistically significant ($p < 0.05$ in each case). The plot in Figure 6.4 shows the average daily balance of the top four agents. Here I have subtracted

Agent	*Score*
TacTex	79.9
AstonTAC	76.2
Schlemazl	75.4
QuakTAC	74.5
munsey	71.8
EPFLAgent	71.7
MetroClick	70.6
UMTac09	66.9

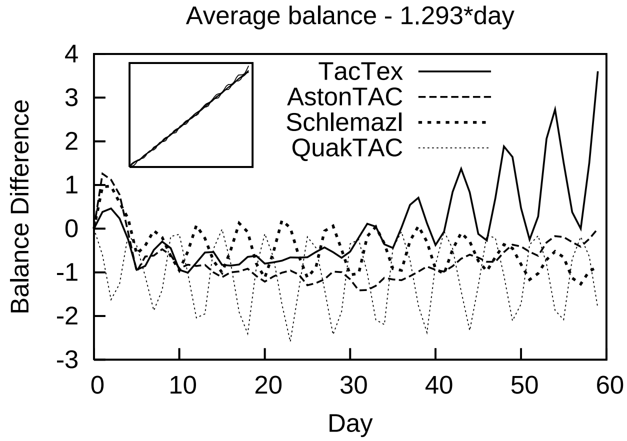

Figure 6.4: 2009 final round scores (in thousands)

1293 per day (the average daily profit of the second place agent) from each balance to improve visibility–otherwise the plot would appear as nearly diagonal lines, as shown in the inset. Two things are apparent from this plot. First, scores tend to oscillate rather than increase smoothly. This oscillation is a consequence of the 5-day capacity window–if an agent has a large number of conversions on one day (frequently the very first day), it will have reduced capacity for four days, and then the heavy day will slide out of the window. There is a similar end game effect, as there is no benefit to saving capacity beyond the last day. TacTex is especially effective at ending on an upswing due to the Multi-day Optimizer. Second, TacTex does not pull away from the other agents until the second half of the game. One possible explanation is that bidding behaviors converge to some degree over the course of a game, and so TacTex's ability to make accurate predictions improves.

Although it is difficult to identify a main reason for TacTex's success, analysis of the game results from the finals does provide a few clues. Compared to other agents, TacTex typically had fewer conversions but had a higher profit per conversion (at least 6% higher than any other agent). AstonTAC and Schlemazl focused on selling those products that matched their manufacturer specialty, which gives a revenue bonus. As a result, their average revenue per conversion was slightly higher that TacTex's, but their clicks tended to come from higher positions and thus their CPCs were much higher than TacTex's. The reverse was true for all other agents: they had lower CPCs, but much lower average revenue per conversion. It appears that TacTex struck the right balance between targeting high revenue conversions and taking advantage of other sales opportunities. A more detailed analysis of the 2009 finals is available in [47].

6.8.2 Experiments

Although our competition victory suggests that the overall design of TacTex is sound, it gives us little information about the relative importance of the individual components. For instance, we cannot look at the results and determine how much the User Model contributed to the overall performance. Fortunately, after the competition, most teams submitted agent binaries to the TAC Agent Repository. [6] Using these binaries, we can experiment by changing certain portions of TacTex and observing the effect on performance. I now report on a number of these experiments.

We measure the impact of a change to TacTex by running two sets of 50 games: one with the original TacTex, and one with a modified version. The other seven agents are the best available agents (as of September 2009) from the repository: AstonTAC, two different versions of Schlemazl, QuakTAC, EPLFAgent, MetroClick, and Merlion. This mixture of agents provides both relatively strong agent performance and a wide variety of behaviors. To improve our ability to evaluate the statistical significance of the results, we have modified the game server so that the main random factors (game parameters, advertiser capacities and specialties, and the random draws primarily responsible for determining user population transitions) are identical between corresponding pairs of games. This allows us to

[6] `http://www.sics.se/tac/showagents.php`

#	*Description*	*Diff.*	*p*
Advertiser Model			
1	bid estimator 1	-284	.150
2	bid estimator 2	-306	.113
3	bids · 0.9	-536	.077
4	bids · 1.1	-678	.010
5	bids · U[0.9,1.1]	-501	.046
6	all impressions	-3438	.000
7	impressions · .9	-772	.008
Other			
8	no User Model	-896	.001
9	no Par. Model	-254	.133
10	no probing	-812	.004

#	*Description*	*Diff.*	*p*
Optimization			
11	maintain cap.	-4257	.000
12	constant conv.	-7362	.000
13	conv. · 0.9	-287	.174
14	conv. · 1.1	+62	.596
15	conv. · 1.2	-1024	.000
16	bids · 0.9	-453	.039
17	bids · 1.1	-704	.000
18	bids · U[0.9,1.1]	-425	.038
19	all generic ads	-2351	.000
20	all targeted ads	-87	.992

Table 6.2: TAC AA Experimental results

use the Wilcoxon matched-pairs signed-ranks test instead of an unpaired test. (A similar approach to modifying the TAC SCM game server was introduced by [94].) Table 6.2 shows the results of our experiments. For each experiment, we give a reference number, a brief description, the difference in score (a negative difference means the score of the modified agent was lower), and the p-value indicating the significance. Experiments are divided into groups corresponding to different parts of TacTex.

The first group of experiments concerns the Advertiser Model. In experiments 1 and 2, we use only one of the bid estimators instead of averaging the two. Although both estimators appear to perform reasonably well on their own, we do see a small increase in score from combining them. In experiments 3-5, we multiply the Advertiser Model's predicted bids for each advertiser by 0.9, 1.1, or a random number chosen uniformly from the range [0.9, 1.1], respectively. Again, we see a small decrease in score from each change. We know from comparison with the true results that our predictions are not always highly accurate, but these experiments show that our predictions are helpful enough that small changes can impact performance negatively. It certainly does not appear to be the case that our predictions are consistently too high or too low. Finally, we look at the Advertiser Model's predictions of the number of impressions advertisers will receive. In experiment 6, we assume that advertisers use no spending limits (i.e., they receive all possible impressions). In experiment 7, when predicting that an advertiser will use a spending limit, we multiply the predicted impressions by 0.9. Both changes result in a large drop in score, showing that impression predictions are very important.

The next group of experiments do not fit a specific category. In experiment 8, we replace the predictions of the User Model with the average user populations expected and see a fairly large drop in score. Similarly, in experiment 9 we replace the estimates from the Parameter Model with the expected values and see a smaller score reduction. In experiment 10, we replace all probe bids with zero bids, and again we see that this is detrimental to performance. The information lost from these changes appears to be important to TacTex, and the last experiment shows that a query type that does not attract attention from TacTex on one day may still become a source of profit in the future.

The last group of experiments concerns optimization. Experiments 11 and 12 show the importance of planning over multiple days. In experiment 11, we choose the next day's conversion target so that the total conversions over the last five days will be 1.4 times our capacity (the average amount used by the unaltered TacTex). Then in experiment 12, we simply set each day's conversion target to be 1/5 of 1.4 times the capacity. Both changes cause an extremely large drop in score despite the fact that TacTex has roughly the same number of conversions. In experiments 13-15, we multiply the next day's conversion target by 0.9, 1.1, and 1.2, respectively. Experiments 13 and 15 result in score reductions as expected; however, multiplying the conversion target by 1.1 actually results in an increase in score, although not a significant one. This result would make sense if TacTex were consistently receiving fewer conversions than expected, but that is not the case. Another possibility is that it may be less harmful to sell too much than too little, and because TacTex will rarely hit its conversion target exactly, it would be safer to target a few extra conversions. In any case, experiments 11-15 suggest that the Multi-day Optimizer is highly effective in choosing conversion targets. Experiments 16-18 concern the bids generated by the Query Analyzer at the end of the optimization process. Again, we multiply these bids by 0.9, 1.1, or a random number chosen uniformly from the range [0.9, 1.1], respectively. In each case there is a moderately large score reduction, and so the Query Analyzer appears to be successful at providing useful information about bid selection. Finally, experiments 19 and 20 concern ad selection. In experiment 19, we use only generic ads, and in experiment 20 we use only targeted ads, where the manufacturer and component match the query when present or TacTex's specialty when null. The results clearly show that ad selection can impact score, but it does not appear that choosing optimal ads (which are usually targeted) is significantly better than simply always using targeted ads.

The main message from these experiments is that nearly all components of TacTex make at least a small contribution to the overall performance of the agent. Some parts, such as the Multi-day Optimizer and impression prediction, appear to be especially important. In addition, from extra experiments not described here we have observed that changing multiple agent components tends to compound the impact on performance. For example, while using only the first bid estimator or multiplying our conversion targets by 0.9 both reduce the score by under 300, doing both at the same time reduces the score by 1238. The true value of some components may therefore be even greater than the experiments shown here suggest.

6.9 2010 Results

In 2010, two significant changes were made to the TAC AA specifications [46]. The first was a drastic increase in reserve prices. The primary result of this change was that agents could afford to focus on only a few query types, causing some query types to receive bids from only one or two advertisers. In 2009, on the other hand, advertisers could place low bids on several query types and get a significant number of low-cost clicks.

The second change was more directly harmful to TacTex. Instead of being given the true average position of each advertiser, agents were given a less precise number that was computed by sampling the positions at ten randomly chosen times. As a result, the Position Analyzer no longer produced accurate estimates of total impressions, bid rankings, and the number of impressions received by other advertisers. This in turn caused the predictions of the User Model and Advertiser Model to be less accurate. In particular, the User Model was no longer nearly as accurate as it was shown to be in Figure 6.2, as TacTex no longer knew the total number of impressions for any query type unless it actually received all these impressions. (Our experiments still showed the User Model to be beneficial, however: removing it resulted in an average score decrease of 435, versus the decrease of 896 seen in the previous experiments under the 2009 specifications.)

Nevertheless, TacTex won the 2010 competition by an even larger margin than in 2009. The scores from the final round are shown in the table in Figure 6.5. One interesting observation from the plot of daily balances is that, unlike in 2009, TacTex began to pull away from the other agents from the start of each game. Our hypothesis is that the increase in reserve prices, and the resulting decrease in competition for each query type, caused agents to settle into predictable bidding patterns earlier on in each game, and so TacTex's advantage in prediction and optimization also appeared earlier.

Agent	*Score*
TacTex	58.1
Schlemazl	52.9
Mertacor	52.4
MetroClick	52.2
Nanda AA	48.1
crocodileagent	47.8
tau	44.4
McCon	43.4

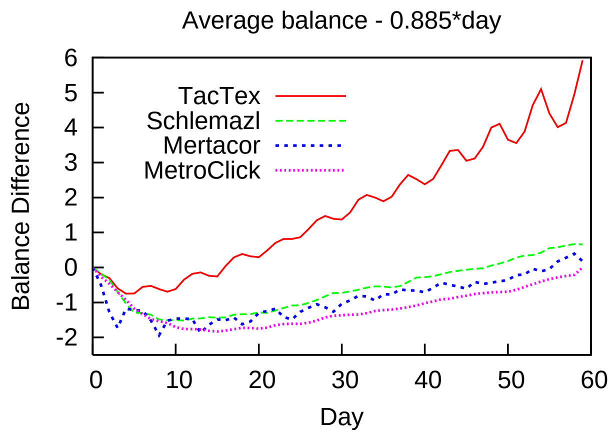

Figure 6.5: 2010 final round scores (in thousands)

6.10 Summary

This chapter described the TacTex agent for the TAC Ad Auctions scenario. TacTex is a complete agent for bidding in keyword auctions. At a high level, it operates by estimating or predicting various quantities and then determining the optimal bids given this information. These tasks are divided among a number of modules, and experiments showed that each module made an important contribution to TacTex's winning performance. As was the case in TAC SCM, after developing a winning agent, the focus of our research agenda became finding ways to apply machine learning to improve the performance of the agent, and the next chapter discusses one such improvement in the area of bid estimation.

Chapter 7

Learning for Bid Estimation in TAC AA

Chapter 6 presented the TacTex agent for TAC AA. The results of the experiments of Section 6.8.2 show that the models used by TacTex for estimation and prediction make an important contribution to its performance. It is thus reasonable to believe that improving the accuracy of these models could improve the performance of TacTex. The User Model and Parameter Model already make use of nearly all available information, and it is unlikely that they could be improved significantly. The Advertiser Model, on the other hand, offers significant room for improvement. In addition, the implementations of the User Model and Parameter Model are in many ways specific to the specifications of the TAC AA scenario, whereas the problem of estimating the bids of other advertisers is one that would appear in a similar fashion in any ad auction setting. For these reasons, our research efforts beyond the initial design of TacTex have focused on the bid estimation problem. This chapter describes an improvement to the competition agent that involves using particle filters for bid estimation. This estimation approach requires the use of machine learning to train bidding models, and so as in Chapter 5, I explore the importance of the choice of training data. This chapter fulfills Contribution 2 for TAC AA, as outlined in Section 1.2.

7.1 An Improved Bid Estimator

The two bid estimators used by TacTex (described in Section 6.5.3) have a number of shortcomings: the first estimator (which I will now refer to as the simple joint estimator) relies on an ad hoc method of updating the particle filter, while the second estimator (which I will now refer to as the simple independent estimator) requires simplifying assumptions to enable bids to be modeled independently and uses a simplistic model of bidder behavior. In this section I present a greatly improved bid estimator, which I will refer to as the full estimator. Like the simple joint estimator, this estimator uses a particle filter to maintain an estimate of the joint distribution over all advertisers' bids. However, the full estimator takes a much more principled approach to updating the particle filter. Like the simple independent estimator, the full estimator uses models of the bidding behavior of other advertisers, but now these models are learned from game data.

As mentioned above, the bid estimation problem we are trying to solve is not specific to TAC AA. In any type of keyword auction, knowing the bids of other advertisers for a specific keyword would allow an advertiser to predict the ad position and cost per click for any amount it bid and use this information to choose the bid it expected to maximize profit. Although search engines typically release some information concerning the position that an advertiser could expect for certain bids, this information is generally incomplete and out of date. Alternatively, an advertiser could experiment with different bids and observe the resulting positions, but such experimentation would be time consuming and costly.

While the specifics of the bid estimation problem can depend on the auction implementation used, at a high level the problem remains the same: estimating the bids of other advertisers given only information about bid rankings. I therefore begin this section by respecifying the bid estimation problem in general terms. I then present the design of the particle filter used by the full estimator. As the particle filter requires bid transition models for all other advertisers, I then describe a machine learning approach to building these models. Finally, I present experimental results comparing the accuracy of the full estimator to other bid estimation methods in three different TAC AA settings.

7.1.1 Auction Setting

We are interested in estimating the bids of N other advertisers for a single keyword for which we are advertising. Each advertiser has a standing bid indicating the amount it is willing to pay each time its ad is clicked, and this bid may occasionally be revised. This bid must be above a known reserve price *reserve*. When a user searches for the keyword, the advertisers' bids are ranked in descending order, and their ads are shown in this order. If more than M bids are above the reserve, then only the top M ads are shown. When a user clicks on our ad, our *cost per click* (*cpc*) is the minimum amount we could have bid and still had our ad shown in its current position. In other words, our *cpc* is equal to the amount of the bid ranked below ours, or to *reserve*. At some regular interval, we receive a report containing the following information: i) the bid ranking at time t (for advertisers whose ads are being shown, i.e., at most the top M), ii) our own *cpc* at time t, and iii) our own *bid* for time t (which of course we already know). Our goal is to estimate the bids of the other advertisers at time t. Depending on the nature of the auction, advertisers may be able to revise their bids more frequently than this reporting interval; however, we will attempt to estimate bids only at this interval, and our models of advertiser behavior will model changes in bids only at this interval (e.g., from time $t-1$ to time t).

As this model is an abstraction of the keyword auctions used in the real world, there are a number of complicating factors it does not include, but we believe it to be a useful model for study. One issue faced in real keyword auctions is that ad positions are often not determined by bid rank alone, but by a combination of bid rank and other factors such as clickthrough rate. This is not a problem, however, as in this case we could simply attempt to estimate the amount we would need to bid to achieve a higher position than each other advertiser, instead of the true bid of each advertiser, and use the same particle filtering approach.

A larger concern is that search engines do not actually provide advertisers with periodic reports of the bid rankings of other advertisers. Of course, it is possible to simply repeatedly search for the keyword and observe the order of the ads displayed, but for a large advertising campaign this process would need to be automated (using a "screen scraper"), and search engines generally take measures to prevent this type of activity. Nevertheless, a number of services offer to collect this type of information for subscribers, so the assumption of these periodic reports is not necessarily unrealistic.

Finally, I note that this auction setting is in fact an instance of a repeated generalized second price auction, and that our particle filter could be applied to any such auction given periodic ranking observations. Generalized second price auctions are most commonly used in keyword auctions, but they have been considered in other areas such as electricity auctions [90].

7.1.2 Particle Filtering

I now describe our particle filter for estimating the bids of other advertisers given periodic reports. For now, I assume that we have a model of each advertiser that gives us a probability distribution over their next bid given a history of their bids and rankings. Developing these models is the subject of Section 7.1.4. Again, I emphasize that we concern ourselves only with the bids at the reporting interval–by "next bid" I mean the bid at the time of the next report, and likewise our history reflects the auction state only at the times of past reports.

Given these advertiser models and reports, we estimate the joint distribution over the bids of all other advertisers using a particle filter. Recall from Section 5.1.1 that a particle filter tracks the changing state of a system by maintaining a set of weighted samples (known as particles) and updating these particles each time a new observation is received. In this case, the reports represent our observations, and each particle represents an estimate of the bids of all advertisers at the time of the last report. Additionally, each particle stores all of its past bid estimates, so each particle can be seen as a full bidding history of all advertisers. Particle filters are a fitting solution to this problem because they require no assumptions about the types of distributions involved (unlike Kalman filters), they can be used efficiently in high-dimensional spaces (unlike grid-based methods that discretize the state space), and particles are a convenient data structure for storing bidding histories. We estimate the joint distribution over bids instead of estimating each advertiser's bid independently due to the fact that our estimate for each advertiser is completely dependent on our estimate for all other advertisers. (Recall that the simple independent filter estimates bids independently but relies on several unrealistic simplifying assumptions.)

For the experiments presented below, the implementation of our particle filter makes use of a discretized set of bids $\{b^1, ..., b^B\}$, and so I describe our particle filter in terms of discrete probability distributions over these bids; however, continuous probability distributions could also be used in our particle filter if they can be dealt with analytically.

7.1.2.1 SIS Particle Filter

The simplest particle filter, and the one from which more complicated variations are derived, is the *Sequential Importance Sampling* (SIS) filter [1]. A SIS filter can be implemented for our bid estimation problem as follows. Each particle p contains a current estimate for the bids of all N advertisers, as well as bid estimates for each past time step. An initial set of particles P is chosen to reflect a possible distribution over bids when no reports have yet been received–essentially our prior. $|P|$ should be chosen to give an acceptable tradeoff between accuracy and speed. Each particle p receives initial weight $w_p = 1/|P|$. Each time we receive a report, we update P by generating and weighting a new set of particles. For each existing particle p, we sample a new particle p' (i.e., we copy the bidding history contained in p and then sample a new set of current bids). Finally, we reweight the particles.

The sampling and weighting procedures depend on our choice of *proposal distribution* from which we sample new particles: $\pi(p'|p, report)$. π may be any distribution we choose. The weighting procedure then follows from the choice of π such that the set of weighted particles approximate the true posterior distribution. If particle p had weight w_p, then particle p' receives weight

$$w_{p'} = w_p \frac{Pr(report|p')Pr(p'|p)}{\pi(p'|p, report)} \ . \tag{7.1}$$

Finally, the weights of all new particles are normalized so that they sum to one.

7.1.2.2 Choice of Proposal Distribution

The choice of proposal distribution can significantly affect the performance of the particle filter. The distribution $\pi(p'|p, report) = Pr(p'|p, report)$ is what is known as the *optimal proposal distribution* and results in a weighting of $w_{p'} = w_p Pr(report|p)$. This proposal distribution is called optimal because it results in the least variance between particle weights–$w_{p'}$ is independent of p', and so it will be the same regardless of which p' is sampled. However, the optimal proposal distribution is often not used in practice because it can be difficult to sample from this distribution and perform weight calculations. Instead, the proposal distribution that would typically be used (which I will refer to as the standard proposal distribution) is $\pi(p'|p, report) = Pr(p'|p)$, which results in a weighting of $w_{p'} = w_p Pr(report|p')$.

The standard proposal distribution is indeed much easier to work with in our bid estimation problem, but is has a serious flaw: $Pr(report|p')$ may frequently be zero. If too few particles receive any weight, then the filter may eventually become degenerate, with mostly identical particles. To see why $Pr(report|p')$ might be zero, recall that the report contains a ranking and our *cpc*. If the current bids represented by p' are inconsistent with this ranking, then the likelihood of p' will be zero. The fraction of inconsistent particles will depend on advertiser behavior;

in the worst case of random bids, only $1/N!$ of the particles would be expected to be consistent with the ranking, as any of the $N!$ possible rankings would be equally likely. Even in less extreme cases, we would still expect there to be occasional improbable rankings. Furthermore, even if p' is consistent with the rankings, it will likely not be consistent with our observed *cpc*. (The simple joint estimator suffers from these same problems and addresses them in an ad-hoc way.)

We therefore use the optimal proposal distribution in our particle filter. Particles drawn from this distribution are guaranteed to be consistent with the report. Thus, we need methods of sampling from $Pr(p'|p, report)$ and computing $Pr(report|p)$. These methods are described in the following two subsections.

7.1.2.3 Computing Pr(report | p)

For $n \in 1...N$, let a_n be the advertiser ranked nth, excluding ourselves (i.e., lower ranked advertisers have their rank increased by one). Unranked advertisers may be assigned to the remaining a values arbitrarily. For a given set of current bid estimates, let c_n indicate that a_n's bid is consistent with the bids of advertisers a_1 ... a_{n-1} and with our own *bid* and *cpc*. Then $Pr(report|p) = Pr(c_1 \cap c_2 \cap ... \cap c_N|p)$. That is, the probability of particle p from the previous time step leading to a new particle consistent with *report* is equal to the probability that for each other advertisers, that advertiser's new bid estimate does not exceed the bid estimate of a higher ranked advertiser or conflict with *bid* or *cpc*. Furthermore, $Pr(c_1 \cap c_2 \cap ... \cap c_N|p) = Pr(c_N|p, c_1 ... \cap c_{N-1}) \cdot ... \cdot Pr(c_2|p, c_1)Pr(c_1|p)$. Below, I show how each of these N probabilities can be computed.

For each $n \in 1 \ ... \ N$, we would like to compute $Pr(c_n|p, c_1 \ ... \ c_{n-1})$, that is, the probability that particle p leads to a new bid for advertiser a_n that is consistent with our *bid* and *cpc* and the new bids of advertisers a_1 ... a_{n-1}, *given that these bids are already known to be consistent.* This probability is computed differently for each of five different cases. Let f_n be the probability mass function for a_n's next bid given the information in p, as determined by our advertiser model for a_n. In each case, we will determine f'_n, the probability mass function for a_n's next bid given p *and* c_1 ... c_{n-1}, as well as the corresponding cumulative distribution function F'_n giving the probability that the new bid is less than (but *not* equal to, as is usual in a CDF) a given value. We begin by setting F'_0 to be 0 everywhere.

- **Case 1:** *a_n has a higher rank than us.* Because the advertiser is ranked, its bid will be consistent with the bids of a_1 ... a_{n-1} so long as its bid is no greater than the bid of a_{n-1}. Because the advertiser is ranked above us, its bid must be no less than *bid*. Therefore,

$$Pr(c_n|p, c_1...c_{n-1}) = \sum_{x=bid}^{b^B} f_n(x)[1 - F'_{n-1}(x)] \tag{7.2}$$

  Similarly, we can define

$$f'_n(x) = f_n(x)[1 - F'_{n-1}(x)]Z \tag{7.3}$$

  where f'_n has support between *bid* and b^B and Z is a normalizing constant.

- **Case 2:** *a_n is ranked one below us.* Our *cpc* is determined by the advertiser ranked below us, so we know the bid of a_n.

$$Pr(c_n|p, c_1...c_{n-1}) = f_n(cpc) \tag{7.4}$$

  and we define F'_n to be 0 at or below *cpc* and 1 elsewhere.

- **Case 3:** *a_n is ranked at least two below us.* As in Case 1, we need the bid of a_n to be no greater than the bid of a_{n-1}. Because the advertiser is ranked below us, its bid must be between *reserve* and *cpc*. Therefore,

$$Pr(c_n|p, c_1...c_{n-1}) = \sum_{x=reserve}^{cpc} f_n(x)[1 - F'_{n-1}(x)] \tag{7.5}$$

and

$$f'_n(x) = f_n(x)[1 - F'_{n-1}(x)]Z \tag{7.6}$$

where f'_n has support between *reserve* and *cpc*.

- **Case 4:** *a_n is unranked and there are M ranked advertisers.* Because the maximum of M advertisers were ranked, we do not know if a_n placed a bid or not. We only know that a_n's bid is no greater than the bid of a_k, where a_k is the advertiser ranked M.

$$Pr(c_n|p, c_1...c_{n-1}) = \sum_{x=0}^{cpc} f_n(x)[1 - F'_k(x)] \tag{7.7}$$

and

$$f'_n(x) = f_n(x)[1 - F'_k(x)]Z \tag{7.8}$$

where f'_n has support between 0 and *cpc* .

- **Case 5:** *a_n is unranked and there are fewer than M ranked advertisers.* a_n did not bid or else it would have been ranked. We treat any non-bid (or bid below the reserve) as a bid of 0, so

$$Pr(c_n|p, c_1...c_{n-1}) = F_n(0) \tag{7.9}$$

and $f'_n(0) = 1$.

By proceeding through the advertisers in order, we can determine $Pr(c_n|p, c_1$ ... $c_{n-1})$ for each $n \in 1$ ... N and take the product to get $Pr(report|p)$. We repeat this process for each $p \in P$ and normalize the results to obtain the distribution from which we sample when generating new particles.

7.1.2.4 Sampling from Pr(p′ | p, report)

Now given a particle p and the report, we would like to sample a new particle p'. This involves choosing a new bid b_n for each advertiser a_n, and so $Pr(p'|p, report) = Pr(b_1 \cap b_2 \cap \ldots\ b_N|p, report) = Pr(b_1|p, report, b_2\ \ldots\ b_N) \cdot \ldots \cdot Pr(b_N|p, report)$.

Observe that for advertiser a_N, the function f'_N generated above is in fact the same as $Pr(b_N|p, report)$ because it represented the distribution over b_N given that the bids of all other advertisers were consistent with *report*. For any other advertiser a_n, if bids $b_{n+1}...b_N$ are known, then we can compute $Pr(b_n|p, report, b_{n+1}...b_N)$ by taking the highest bid of any lower ranked advertiser (if any) and normalizing the portion of f'_n above that bid. Thus, by starting with b_N and working backwards, we can sample all bids in such a way that the bids are consistent with *report* and the probability of the resulting particle p' is $Pr(p'|p, report)$.

7.1.2.5 Example

I now use an example to illustrate particle filters using both the standard and optimal proposal distributions. Suppose that there are two advertisers x and y in addition to ourselves, and that according to our advertiser models, at each time step each either increases or decrease its bid by 1, with probability 0.5 in each case. We receive a report for time $t + 1$ indicating that y had the highest bid, x had the second bid, and we had the lowest bid of 0.25. Now consider a particle p that has the following bid estimates for time t: $b_x = 2$ and $b_y = 1.5$.

For the standard proposal distribution, to sample a new particle p' reflecting time $t + 1$ we would sample new bids for each bidder according to our advertiser models. However, of the four possible outcomes, only one, $b_x = 1$ and $b_y = 2.5$, is consistent with the bid ranking. If we sampled a different set of bids for p', then the weight of p' would be set to zero.

For the optimal proposal distribution, we let $a_1 = y$ and $a_2 = x$ and follow the procedure described above. First, we determine $Pr(report|p)$. We have $f_1(0.5) = f_1(2.5) = 0.5$ and $f_2(1) = f_2(3) = 0.5$, with both functions zero elsewhere. For a_1, we follow Case 1. $Pr(c_1|p) = 1$ and $f'_1 = f_1$ because F'_0 is zero and either possible bid is above our bid of 0.25. For a_2, we follow Case 1 again. $Pr(c_2|p, c_1) = 0.25$ and $f'_2(1) = 1$, because $1 - F'_1(3) = 0$, $1 - F'_1(1) = 0.5$, and either possible bid is above our bid of 0.25. Thus $Pr(report|p) = Pr(c_2|p, c_1)Pr(c_1|p) = 0.25$, which we know is correct.

To sample a new particle p', we first sample b_2 from f'_2 and get 1, the only possibility. Then we sample b_1 from the portion of f'_1 that is above 1, and we get 2.5, again the only possibility. So we are guaranteed to sample $b_1 = 2.5$ and $b_2 = 1$, the only possibility for p' given *report*.

7.1.2.6 Resampling

The particle filter implementation described so far has been an SIS particle filter using the optimal proposal distribution. A commonly used extension of an SIS filter is the *Sampling Importance Resampling* (SIR) filter, which occasionally resamples the set of particles to prevent the weights of some particles from approaching zero. Our implemented particle filter is an SIR filter, and we resample the particles in P after each update by replacing P with $|P|$ particles sampled (with replacement) according to the weights, then setting all weights to $1/|P|$. The full procedure for updating the particle filter is summarized in Algorithm 8.

7.1.3 Application to TAC AA

TAC AA differs from the auction model described in Section 7.1.1 in a number of ways. First, the daily reports provide the average positions of other advertisers instead of a ranking of their bids. Fortunately, as described in Sec-

Algorithm 8: UPDATE-FULL-ESTIMATOR

```
begin
    input : report containing a ranking, our bid, and our cpc.
    for each particle p do
        sort advertisers by ranking
        for each advertiser a_n do
            compute f_n, the probability mass function over a_n's next
            bid, using a_n's advertiser model
            set weight_p to Pr(report|p), computed as in Section 7.1.2.3,
            and store resulting functions f'_n
    normalize particle weights
    resample particles, and set each weight to 1/|P|
    for each particle p do
        sample a new particle p' by sampling a bid for each advertiser
        as in Section 7.1.2.4
```

tion 6.3 it is possible to transform average positions into bid rankings with fairly high accuracy. Second, in TAC AA the reserve price is unknown, but we can obtain a reasonably accurate estimate and use this in our particle filter. Third, we are only given an average cpc. If the agent ranked one spot below us hits its spending limit before we do, the average cpc will not equal the bid of that agent, as was assumed in Case 2 above. Once again, we can use the reported average positions to determine if this was the case, and if so we can apply Case 3 instead for that advertiser. Finally, in TAC AA ad positions are determined by a combination of bid rankings and clickthrough rates. In our experiments, we address this issue by adjusting each bid of each other advertiser to be the amount we would have needed to bid to achieve a higher position than that advertiser.

The use of spending limits in general represents another significant difference. We avoid dealing with spending limits by using our particle filter to estimate each advertiser's bid at the start of the day, before spending limits cause any advertiser to drop out of the bidding. Estimating the spending limits of other advertisers can be treated as a separate problem, as described in Section 6.5.1, although estimating bids and spending limits together is a possible area for future work.

The application of our particle filter to a single query type in a TAC AA game can therefore be summarized as follows. On each day d, we receive a report that includes our average cpc and the average position of all advertisers on day $d-1$. We transform the average positions into the bid rankings, and we determine whether our average cpc does in fact equal the bid of the advertiser ranked below us. Then, using this information, we update our particle filter, and the result is an estimated distribution over the bids of all advertisers at the start of day $d-1$.

In our experiments, without loss of generality we consider only the nine F2 query types–those in which both a manufacturer and component are specified. In any one game, a number of random factors affect the value of advertising for any particular query type, and thus the bidding behavior of the agents. The distributions from which these factors are drawn are the same for all nine query types, however, and so this bidding behavior is the same in expectation for any query type in any game. As our particle filter is designed for a single keyword auction, in our experiments we treat each game as if it provides us with nine independent and identically distributed episodes, each representing a 60-day bidding history for a single keyword.

7.1.4 Advertiser Models

In Section 7.1.2, I assumed that we had a bidding model for each advertiser so that we could determine the distribution over the advertiser's next bid given its bid history. I now describe a method of generating such models using machine learning. While the details of this section are specific to TAC AA, the general approach could be used in any situation in which sufficient bidding data is available for use in learning. For a given advertiser, we assume that we have access to the logs from a number of TAC AA games in which both that advertiser and our own agent participated. From these logs, we can determine the actual bids of the advertiser as well as the reports that would have been available to our own agent at any point in time. Such complete data would not be available in real keyword auctions, of course, but some information is typically provided about past bids. In addition, it might be possible to obtain sufficiently accurate models using a bootstrapping process, i.e., alternating between training models and then using them to estimate past bids.

The problem we are trying to solve is a conditional density estimation problem. While a number of parametric approaches to solving these problems exist, we choose to use a nonparametric approach. The bidding behavior of advertisers can be quite complex, and we would prefer to make as few assumptions about this behavior as possible. Methods of nonparametric conditional density estimation have been used in previous TAC domains to solve problems such as predicting future hotel prices [88] and predicting the probability of an offer to a customer resulting in an order [55]. The approach we take is to learn a model that takes as input both a bid amount b and a set of features representing the current state, and outputs the probability that the advertiser's next bid is less than or equal to b. Thus by evaluating this model for different values of b, we can build the cumulative distribution function for the advertiser's next bid for any given state. This approach is similar to the one used in [88] except that rather than including a price as an input to the model, there the space of prices is discretized and the model outputs a separate probability prediction for each price.

For each day $d > 0$ and any given bid b for which we wish to make a prediction we generate a feature vector containing the following:

- b,
- d,

- the last five bids: b_{d-1} ... b_{d-5},
- five bid differences: $b - b_{d-1}$ ... $b - b_{d-5}$,
- the last average position: ap_{d-1},
- five average position differences: $ap_{d-1} - ap_{d-2}$ ... $ap_{d-1} - ap_{d-6}$,
- the maximum and minimum bids so far: max and min,
- the differences $b - max$ and $b - min$,
- the maximum and minimum bids over the last ten days: max_{10} and min_{10}, and
- the differences $b - max_{10}$ and $b - min_{10}$

Any reference to a day before the first day is replaced with the corresponding reference to the first day. Finally, each vector is labeled with a 1 or 0 to indicate whether the advertiser's bid on day d was less than or equal to b.

Observe that any choice of b results in a unique feature vector. To generate a set of training data from game logs, we need to choose one or more values of b to use for each bid observed. If on day d the advertiser's bid was b_d, we generate 14 training instances by using 14 different values of b. The first two values are b_d and $b_d + 0.01$. The next two values are 0 and $\hat{b}$, where $\hat{b}$ is 1.1 times the highest bid ever observed for the advertiser. We then generate ten random bid values between 0 and $\hat{b}$, chosen to give good coverage of the range of possible bids. The number 14 was chosen to give a reasonable tradeoff between model accuracy and keeping the size of the training set manageable.

Now that we have a training set, we need to choose a learning algorithm to build our model. We experimented with the learning algorithms available in the WEKA machine learning toolkit [110] and found that M5P model trees gave the best performance both in terms of probability prediction accuracy on the data set and bid estimation accuracy of the particle filter. Note that our learning problem can be treated as either a regression problem (treating the probability as a number to predict) or a binary classification problem (predicting the probability of belonging to the class '1'), and so both types of algorithms were tested.

One final issue that must be dealt with is the fact that we wish to use our model to produce a cumulative distribution function, but the output of our model may not in fact satisfy the requirements. For a given state, as the bid b increases from 0 to $\hat{b}$, the output of our model should monotonically increase and reach a maximum of 1, but this will sometimes not be the case. We address this problem as follows. Let the function $g(b)$ represent the output of our model for bid b in the current state. In our particle filter, we work with a discretized set of bids b^1 ... b^B. We define a probability mass function f over this set of bids:

$$f(b^i) = max(g(\frac{b^i + b^{i+1}}{2}) - g(\frac{b^i + b^{i-1}}{2}), \epsilon)Z \tag{7.10}$$

for a normalizing constant Z and some small $\epsilon > 0$. The corresponding cumulative distribution function F is now strictly increasing, and $F(b^B) = 1$.

Each time the particle filter needs to draw a new particle p' based on an existing particle p, we generate f and F for each advertiser and then follow the procedure described in Section 7.1.2.1. Note that in this case, the advertiser's bid history used to generate the feature vector that is input to the model is based on the bid history stored in the particle p, and not on the (unknown) true bid history.

7.1.5 Experiments

I now report on experiments that demonstrate the effectiveness of our particle filter (the full estimator) for bid estimation. I begin by presenting the experimental setup and describing the alternate bid estimation methods against which we compare the full estimator.

7.1.5.1 Setup

We evaluate the full bid estimator in three different settings. For each setting, we use TacTex as the advertiser we participate as (i.e., the agent whose observations we see and on whose behalf we are estimating bids). The other seven advertiser agents are chosen from the TAC Agent Repository. We use agents and the game server from 2009. Different sets of agents are used for each of the three settings. For each setting, we run 50 games. 40 games are used to generate training data, and the remaining 10 are used for testing. For each advertiser, we train a model as described in Section 7.1.4.

For testing, we run our estimator independently for each of the 90 60-day bidding episodes (nine per game, as described in Section 7.1.3) contained in the test games. In each episode, we initialize each particle by drawing a bid for each advertiser from a histogram of that advertiser's initial bids. Then each day, we update our bid estimates by giving the particle filter the bid rankings, average positions, and TacTex's bid and cpc for that day and performing the update procedure described in Section 7.1.2.1.

In our particle filter implementation, we use 2000 particles and did not observe an increase in accuracy from increasing this number. We discretize the bid space into intervals of 0.01, with a maximum bid of 1.1 times the highest bid observed from any advertiser, and we set $\epsilon = 0.0001$.

Our goal in estimating bids is not to track the behavior of a specific advertiser but to get an idea of how much we would need to bid to reach a certain position. We therefore evaluate the performance of our estimator by comparing our estimate of the nth ranked bid (based on the weighted mean of the particles) with the actual bid for each relevant value of n.

The first setting we consider involves a set of seven different advertiser agents: AstonTAC, QuakTAC, epflagent, MetroClick, Merlion, and two different

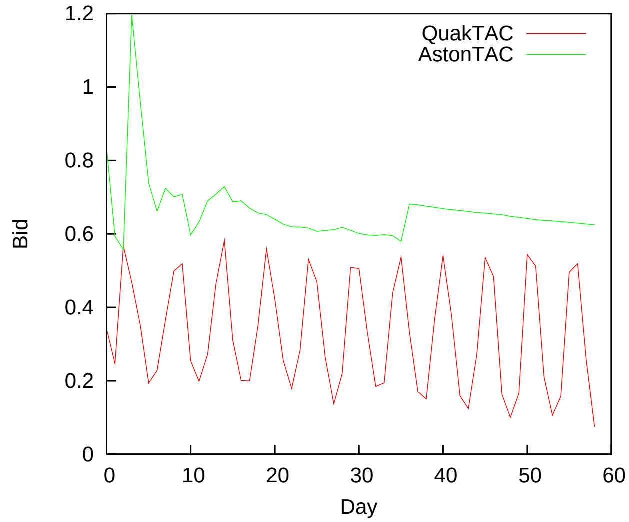

Figure 7.1: Daily bids of two advertisers

versions of Schlemazl (Schlemazl1 from the semifinals and Schlemazl2 from the finals). Bidding strategies differ considerably between agents. For example, Figure 7.1 shows the bids of AstonTAC and QuakTAC for one particular episode. Here AstonTAC's bids tend to drift only slightly from day to day, while QuakTAC's bids take larger jumps but show a clear cyclical pattern. Figure 7.2 shows how the top five bids change each day, illustrating the difficulty of the bid estimation problem. For our second setting we run TacTex against seven copies of QuakTAC, and for our third setting we run TacTex against seven copies of AstonTAC.

To evaluate the effectiveness of the full estimator, we need to compare its accuracy to other bid estimation methods. The first method we consider is a simple baseline of always estimating the *nth* ranked bid to be the average *nth* bid over the training set. We also compare the full estimator against the simple joint estimator and simple independent estimator. Finally, we test the full estimator using the the simple bidding model from the simple independent estimator for all advertisers. Recall that this model assumes that bids either jump randomly, change slightly from the previous bid (as with AstonTAC), or change slightly from the bid 5 days ago (as

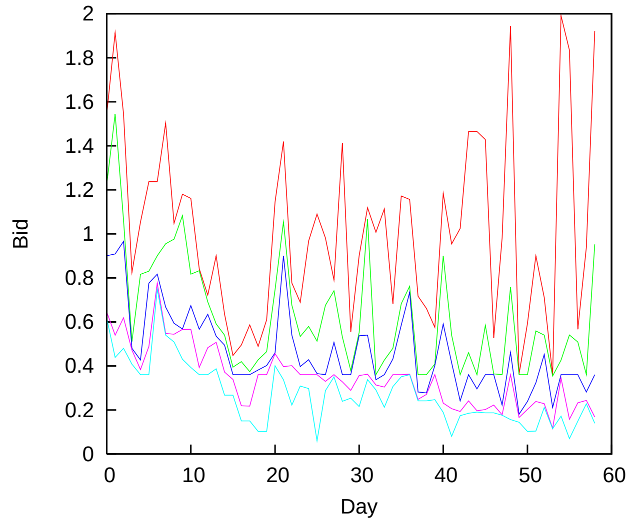

Figure 7.2: Top five daily bids

with QuakTAC). We also considered the opposite combination–using the bidding models described in Section 7.1.4 with the simple independent estimator. However, because the simple independent estimator maintains only bid distributions, and not particles representing bid histories, we do not have the information required to use these bidding models. We tried using the mean of each advertiser's bid on each previous day as the bid history, but results were poor. We also tested a version of the simple independent estimator based on a particle filter using the standard proposal distribution, instead of a grid-based estimator, but it suffered from degeneracy problems similar to those of a joint-distribution particle filter when using the standard proposal distribution (as described in Section 7.1.2.2).

7.1.6 Estimation Results

Tables 7.1, 7.2, and 7.3 show the results for all bid estimators for settings 1, 2, and 3, respectively. For each of the 90 episodes from the 10 test games, we found the root mean squared error of the estimates, and the average RMS error is displayed. We ignored the first five game days in computing these errors so that the errors would not be skewed by start-game effects. (The method of simply using the average bid was especially inaccurate during this period.) In settings 1 and 2, the errors of the estimates for the top five bids are shown, since there were nearly always at least five ranked bidders in these settings. In setting 3, however, there were often only three ranked bidders, and so three errors are shown.

For all bid estimators, errors were highest on the top ranked bid and generally decreased as the rank increased. This result is expected since the top bid is essentially unbounded above and can fluctuate significantly (as in Figure 7.2), while lower bids tend to be grouped more tightly. Errors on settings 2 and 3 were much lower than on setting 1. Both QuakTAC and AstonTAC have somewhat predictable bidding patterns and avoid particularly high bids.

The full estimator with the learned bidder models consistently gave the lowest error of any estimator. In all but one case (setting 3 rank 3) the difference between this error and all other errors was statistically significant ($p < 0.05$) according to a Wilcoxon matched-pairs signed-ranks test. The simple joint estimator had the highest errors overall, although it appears that occasional very high errors (often early in games) were responsible for much of this outcome. Not surprisingly, simply using the average bid also performed poorly. The performance of the full estimator using the simple advertiser model and the simple independent estimator (which uses the same model) was mixed, with neither clearly outperforming the other. This result is somewhat surprising, since the full estimator is in theory a more principled approach. It may be the case that the deficiencies of the simple advertiser model affect each approach differently and that in some cases the simple independent estimator is more robust.

Bid Estimator	*Average RMS error per bid estimate for each bid rank*				
	1	2	3	4	5
average bid	0.678	0.302	0.228	0.176	0.155
simple joint	0.798	0.336	0.279	0.264	0.272
simple independent	0.685	0.289	0.190	0.119	0.110
full, simple model	0.603	0.304	0.187	0.115	0.089
full, learned models	**0.459**	**0.255**	**0.135**	**0.082**	**0.066**

Table 7.1: Bid estimate errors for all estimators - setting 1. Significantly lowest errors in bold.

Bid Estimator	*Average RMS error per bid estimate for each bid rank*				
	1	2	3	4	5
average bid	0.191	0.135	0.106	0.086	0.079
simple joint	0.214	0.126	0.123	0.131	0.142
simple independent	0.203	0.110	0.096	0.085	0.095
full, simple model	0.163	0.089	0.080	0.098	0.118
full, learned models	**0.112**	**0.060**	**0.055**	**0.049**	**0.052**

Table 7.2: Bid estimate errors for all estimators - setting 2. Significantly lowest errors in bold.

Bid Estimator	*Average RMS error per bid estimate for each bid rank*		
	1	2	3
average bid	0.234	0.127	0.178
simple joint	0.233	0.240	0.202
simple independent	0.185	0.198	0.185
full, simple model	0.206	0.127	0.095
full, learned models	**0.135**	**0.102**	0.092

Table 7.3: Bid estimate errors for all estimators - setting 3. Significantly lowest errors in bold.

7.1.7 Application to Bidding

Finally, while the focus of this section has been on estimating bids accurately, the goal of this estimation is ultimately to allow an advertiser to set its own bids effectively. I now briefly explore the usefulness of our particle filter when utilized by a full bidding agent. For each setting, we ran 50 games using the original TacTex agent with the simple independent estimator, then repeated these games using the full estimator with the learned bidder models. Surprisingly, TacTex's score did not improve in setting 1, apparently due to issues with the estimation of other advertisers' spending limits that we have been unable to resolve. (It may be the case that bids and spending limits need to be estimated jointly.) Fortunately, QuakTAC and AstonTAC do not make significant use of spending limits, so this problem does not impact settings 2 and 3. In setting 2, using the full estimator improved TacTex's score by 452 (from 78,177), and in setting 3, the score improved by 926 (from 82,424). In both cases, the increase was statistically significant ($p < 0.05$) according to a Wilcoxon matched-pairs signed-ranks test. For reference, we ran each set of games again and fed TacTex the true bids of the other advertisers, and the scores in settings 2 and 3 increased by 847 and 1582, respectively, compared to the scores when the simple independent estimator was used. Thus, the use of the full estimator appears to provide us with a large portion of the gain to be had from improving bid estimation accuracy.

7.2 Learning Advertiser Models

The previous section introduced a particle-filter-based approach to estimating the bids of other advertisers in keyword auctions given a periodic ranking of their bids. This estimator makes use of models of other advertisers that have been trained on past bidding data. While the estimator was shown to be more accurate than other approaches, the accuracy is clearly influenced by the accuracy of the advertiser models. As was the case for TAC SCM in Chapter 5, it is important to consider what sources of data would be available for an agent to use in training these models.

One important distinction between the problem of advertiser modeling addressed in this chapter and the price prediction problems faced in TAC SCM is that we are modeling each agent (i.e., advertiser) individually, instead of modeling the market as a whole. Still, we face a similar decision about how to train models. In a real keyword auction, if a new advertiser begins bidding on a particular keyword, we have three possible sources of training data: the limited data for that advertiser and keyword, data for other advertisers that have been bidding on that keyword, or, if that advertiser has been bidding on other keywords, that data. The latter two data sources are examples of the previous market experience described in the introduction, and as outlined in Section 1.1, there are several approaches that can be taken to using this data. In this section I consider the two simplest approaches: learning online from new data, or learning from previous experience only. Training online should be sufficient if enough data is available. Before that point, it might make sense to train on data for that advertiser for other keywords, but only if we can be sure the advertiser's behavior is similar between keywords. Likewise, the accuracy of a model trained on data for other advertisers depends on the similarity between these advertisers and the new advertiser.

In the context of a TAC AA tournament, we face similar choices, and I explore these choices in this section. For the purposes of this section, I will assume that game logs are available immediately after each game, although that is not the case during competition. I again use the three settings from the previous section: sets of 40 games in which TacTex plays against either 1) seven different agents, 2) seven copies of QuakTAC, or 3) seven copies of AstonTAC. For a particular agent, we could train online on data from that set as it becomes available. For setting 1, we could train on data from another agent. (For simplicity I assume that all game data for other agents is available at all times, even if only data from some games is available for the agent of interest. This simulates the case in which one new agent begins bidding on a keyword; alternately, we could pretend that the data for the other agents came from a previous round of competition.) Finally, for QuakTAC and AstonTAC, we could train using data for that agent from a different setting, since each agent participates in two settings. (In experiments involving settings 2 and 3, I use data from and model only one of the seven agents, since we would not normally know that all agents used the same strategy; however, once a model is learned, it can be used for all seven agents.)

7.2.1 Generalization

I begin by exploring how well models generalize from one agent to another, including the case of the same agent in different settings. For each setting and agent (including TacTex), models are trained on the 40 training games from Section 7.1.5.1. Models are then evaluated on the 10 test games of all settings and agents. The simple model described in Section 6.5.3.2 is also evaluated. Table 7.4 shows the error of each model on each test data set. Here the agents in settings 2 and 3 are labeled as QuakTAC7 and AstonTAC7, respectively. The lowest error on each data set is shown in bold, and in each case this result is statistically significant ($p < 0.05$) according to a Wilcoxon matched-pairs signed-ranks test.

Several observations can be made from the results. As expected, for each agent, the model trained for that agent had the lowest error of any model. Errors were often much higher when other models were used, indicating that bidding behavior can differ significantly between agents. For instance, the model trained for QuakTAC does poorly when tested on AstonTAC, and vice-versa, as we would expect given the bidding patterns seen in Figure 7.1. The simple model fared reasonably well in these tests. Looking at each data set, the simple model was rarely among the least accurate models, and in several cases it was among the most accurate, suggesting that the simple model indeed serves as a reasonable baseline for comparison in our experiments. Bidding patterns are clearly more predictable for some agents than others. MetroClick appears to be the most predictable agent

	Test Data									
Model	*epfl*	*Merl*	*Metro*	*Sch1*	*Sch2*	*TT*	*Ast*	*Quak*	*Ast7*	*Quak7*
epfl	**.1957**	.3678	.3080	.4553	.3829	.4443	.1563	.3209	.1635	.3194
Merl	.2629	**.2605**	.2958	.4179	.3505	.3907	.1746	.3160	.1775	.3109
Metro	.4983	.5164	**.0355**	.6220	.5853	.5496	.5007	.5007	.5100	.5114
Sch1	.2473	.3217	.2882	**.3680**	.3318	.3781	.2374	.3082	.2204	.3088
Sch2	.2625	.3373	.3028	.3826	**.2983**	.3830	.2264	.2994	.2119	.3003
TT	.2565	.3500	.2635	.3971	.3468	**.3475**	.2532	.2646	.2501	.2736
Ast	.2352	.3464	.2887	.4559	.3807	.4486	**.1117**	.3382	.1209	.3379
Quak	.2742	.4006	.3195	.4765	.3950	.4343	.2308	**.2135**	.2179	.2277
Ast7	.2271	.3543	.2903	.4545	.3720	.4486	.1163	.3459	**.1158**	.3472
Quak7	.2757	.4000	.3375	.4690	.4132	.4271	.2876	.2204	.2782	**.2164**
simple	.2162	.3326	.2671	.4086	.3742	.3883	.2776	.2498	.2778	.2553

Table 7.4: RMS error when models are learned for one agent and tested on another

by far (i.e., it has the lowest self-error), and in fact it turns out that the version of MetroClick used in these experiments changes its bid very rarely. Schlemazl1 is the most unpredictable agent, and its bids indeed appear very erratic. Finally, looking at the errors for QuakTAC and QuakTAC7, as well as AstonTAC and AstonTAC7, we see that for both agents, training on data obtained for that agent under a different setting (i.e., group of opponents) produces a model that is nearly as accurate as training on data from the same setting. This result is promising, as it suggests that an agent's behavior is not highly dependent on the agents it is bidding against, and thus training on data from a different round of competition (or on a different keyword in real auctions) might be an effective strategy.

To better visualize the data in Table 7.4, we can perform agglomerative (bottom up) clustering using the distance function defined in Section 5.4.1. The resulting dendrogram is shown in Figure 7.3. This dendrogram helps illustrate which agents are most similar to each other in bidding behavior. For example, AstonTAC and epflagent are fairly similar, and TacTex, Schlemazl1, and Schlemazl2 are also similar to each other. MetroClick is significantly different from any other agent.

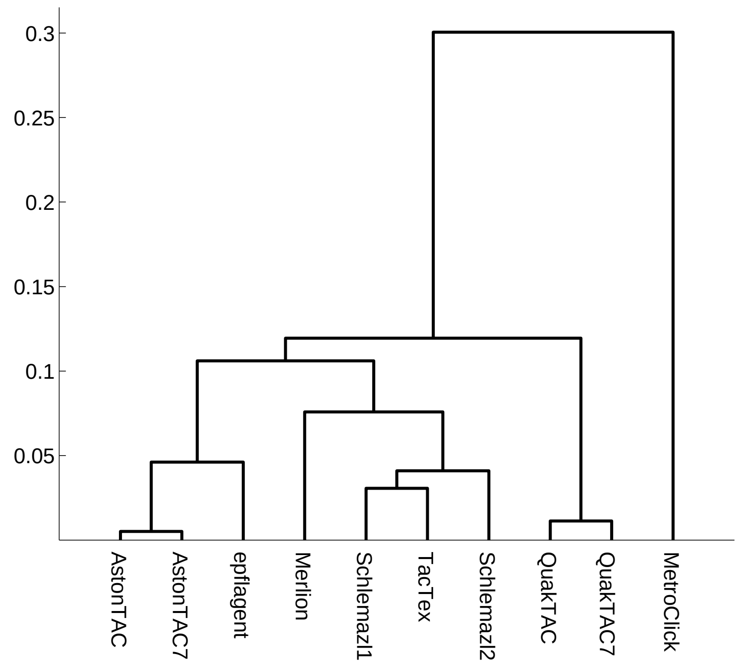

Figure 7.3: Dendrogram of bid model similarity based on Table 7.4

7.2.2 Learning Online

The alternative to training on data for a different agent, or for the same agent under different circumstances, is learning online as data becomes available. For QuakTAC, AstonTAC, and Schlemazl1 (the most unpredictable agent), we generated learning curves by training models on 30 different random subsets of the training games for different numbers of games and testing on the test games. The average RMS errors are shown in Figures 7.4, 7.5, and 7.6, along with the errors of the most accurate other models.

As was the case in TAC SCM, the decision of whether to learn online or to use fixed models trained on a different data set is not clear cut. For AstonTAC and QuakTAC, if we have data for the same agent in a different setting, it takes about 12 games for the accuracy of the online model to reach that of the other model. If data from a different setting is unavailable, however, then training online

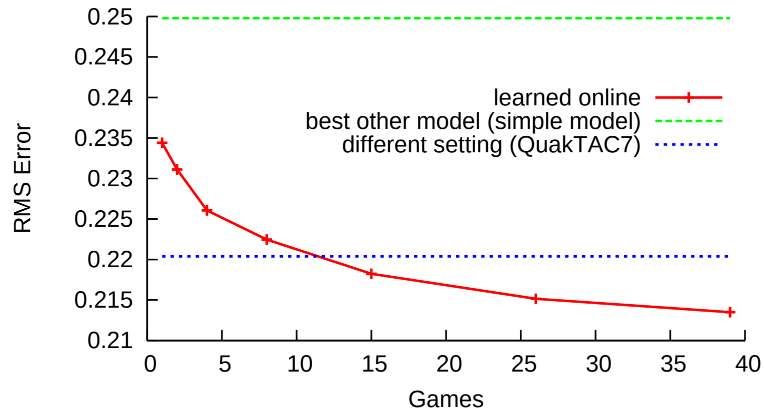

Figure 7.4: Bid model error for QuakTAC when learned online

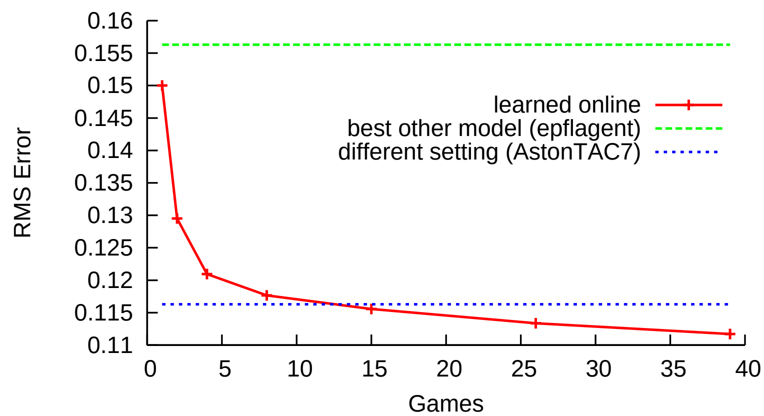

Figure 7.5: Bid model error for AstonTAC when learned online

beats the best model trained on another agent from the same setting (or the simple model) after only one game. For Schlemazl1, it takes about eight games for the accuracy of the online model to reach that of the best other model. Overall, if forced to choose between learning online or using a fixed model, it appears that learning online would be the better choice unless we know that another agent has very similar behavior or we have data for the same agent in a different setting. A more appealing possibility is the idea of somehow combining the two options, as outlined in Section 1.1 of the introduction, and this is the topic of Chapters 8 and 9.

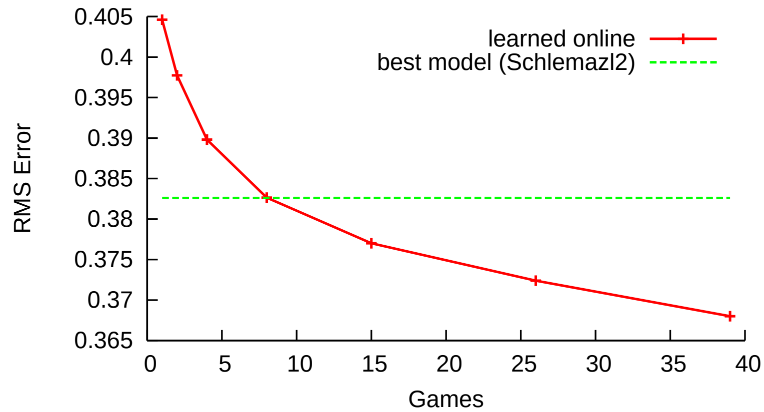

Figure 7.6: Bid model error for Schlemazl1 when learned online

7.2.3 Estimation Accuracy

As the purpose of the advertiser models is to enable us to estimate advertisers' bids, it is reasonable to examine whether improvements in model accuracy translate to improvements in estimation accuracy. The experiments of Section 7.1.6 showed that when used with the full estimator, models trained on 40 games resulted in more accurate estimations than the simple model. To better gauge the importance of model accuracy, we repeated those experiments with additional models for settings 2 (QuakTAC7) and 3 (AstonTAC7), as these settings require only a single model to be chosen. In addition to the simple model and the model trained on 40 games, we tested a model trained on a different agent (AstonTAC or QuakTAC, respectively), a model trained on the same agent from setting 1 (QuakTAC or AstonTAC, respectively), and a model trained on only eight games from the same setting.

Tables 7.5 and 7.6 show the results on the set of 10 testing games for each setting, including the error of each model on the corresponding data set and the bid estimate errors of the full estimator using each model. Although the correlation is not perfect, more accurate models typically result in lower bid estimation errors, and the model trained on all 40 games always resulted in the lowest errors. As

Bid Model	*RMS Error on data set*	*Avg. RMS error per bid estimate for each rank*				
		1	2	3	4	5
simple	0.2553	0.163	0.089	0.080	0.098	0.118
AstonTAC	0.3382	0.159	0.091	0.078	0.080	0.087
QuakTAC	0.2277	0.146	0.067	0.061	0.053	0.056
8 games	0.2274	0.128	0.062	0.058	0.054	0.056
40 games	0.2164	0.112	0.060	0.055	0.049	0.052

Table 7.5: Bid estimate errors using different models - setting 2 (QuakTAC7)

Bid Model	*RMS Error on data set*	*Avg. RMS error per bid estimate for each rank*		
		1	2	3
simple	0.2776	0.206	0.127	0.095
QuakTAC	0.3382	0.258	0.173	0.118
AstonTAC	0.1209	0.151	0.108	0.097
8 games	0.1211	0.144	0.124	0.114
40 games	0.1158	0.135	0.102	0.092

Table 7.6: Bid estimate errors using different models - setting 3 (AstonTAC7)

a result, efforts to improve model accuracy appear likely to lead to better bid estimates, and hopefully better agent performance.

To further examine the effect of model accuracy on agent performance, we repeated the experiments of Section 7.1.7 in which we replayed 50 games using various bid estimation methods. For setting 2, using the full estimator with the model trained on AstonTAC instead of QuakTAC7 (all 40 games) resulted in a decrease in score of 780. For setting 3, using the full estimator with the model trained on QuakTAC instead of AstonTAC7 (all 40 games) resulted in a decrease in score of 1687. For both settings, the decrease was statistically significant ($p < 0.05$) according to a Wilcoxon matched-pairs signed-ranks test. Thus, model accuracy can indeed affect agent performance.

7.3 Summary

A particularly important problem, both within TAC AA and in real keyword auctions, is estimating the bid that would be required to obtain a particular ranking. This chapter presented a method of bid estimation based on using a particle filter to maintain a joint distribution over the bids of all other advertisers. The implementation of this particle filter depends on a method of working with the optimal proposal distribution, which I also described. Experiments show this bid estimation method to produce more accurate estimates than other methods considered, including the two methods originally used by TacTex during the TAC AA competition.

The bid estimation method depends on models of other advertisers. Each model predicts the probability of a given advertiser submitting a particular bid given its bidding history. These models are trained using machine learning on past bidding data. The experiments of this chapter show that the accuracy of the bid estimator depends heavily on the accuracy of these models. As a result, the decision of how to train these models is an important one. As was the case for TAC SCM in Chapter 5, the decision of whether to train models online or use fixed models trained on another source of data (other advertisers, in this case) depends on the amount of training data available and the similarity of agents. In both SCM and AA, we wish to train a model for regression given one highly relevant but small data set and other data sets that are potentially less relevant but much larger. The following chapter explores algorithms for solving this problem.

Chapter 8

Transfer Learning for Regression

Chapters 4 through 7 described the TacTex agents for SCM and AA and showed the importance of learning models of other market participants. In both agents, supervised learning was used to solve regression problems using experience from a specific source: either the market in which the agent is currently participating, or some other market for which a larger amount of data was available. I now consider the general problem of performing regression when multiple sources of data are available that represent different concepts: an instance of *Transfer Learning*. In this chapter I introduce a number of algorithms for regression transfer. I then present experimental results for these algorithms in a wide variety of test domains, including the TAC domains from previous chapters. This chapter fulfills Contribution 3 for both TAC scenarios, as outlined in Section 1.2. A full exploration of the use of these algorithms within TAC is left to Chapter 9. A particularly important contribution of this chapter is Two-stage TrAdaBoost.R2, a novel regression transfer algorithm that is consistently among the best-performing algorithms across the test domains considered.

8.1 Problem Specification

The idea behind transfer learning [74] is that it is easier to learn a new concept (such as how to play the trombone) if you are already familiar with a similar concept (such as playing the trumpet). In the context of supervised learning, inductive transfer learning is often framed as the problem of learning a concept of interest, called the *target concept*, given data from multiple sources: a typically small amount of *target data* that reflects the target concept, and a larger amount of *source data* that reflects one or more different, but possibly related, *source concepts*.

Formally, our goal is to learn a model of a concept c_{target} mapping feature vectors from the space X to labels in the space Y. In binary classification problems, $Y = \{0, 1\}$, while in the regression problems studied here, $Y = \mathbb{R}$. We are given a set of training instances $T_{target} = \{(x_i, y_i)\}$, with $x_i \in X$ and $y_i \in Y$ for $1 \leq i \leq n$, that reflect c_{target}. In addition, we are given data sets $T^1_{source}, \ldots, T^B_{source}$ reflecting B different but related concepts also mapping X to Y. In order to learn the most accurate possible model of c_{target}, we must decide how to use both the target and source data sets. If T_{target} is sufficiently large, we can likely learn a good model using only this data. However, if T_{target} is small and one or more of the source concepts is similar to c_{target}, then we may be able to use the source data to improve our model.

8.2 Algorithms

A number of algorithms have been developed to address inductive transfer in classification settings, but much less attention has been paid to regression settings. In this section, I describe ten algorithms that can be used for regression transfer, including existing algorithms from the literature, modifications of existing classification-specific algorithms for regression, and new algorithms.

These algorithms can be divided into the categories presented in Section 1.1 of the introduction. The first two algorithms correspond to the first two categories: learning from target data only, and identifying the most similar source. The remaining algorithms belong to the final category: learning from both source and target. These algorithms can be subdivided into two groups: algorithms that use the source data directly, and algorithms that make use of models representing source concepts (which can thus be seen as ensemble methods). This second group of algorithms could either be given the source models directly or be given source data and train these models as an initial step, and we use the latter approach in our experiments. In selecting the algorithms explored in this chapter, we focused on algorithms that can be used in conjunction with any regression base learner. This decision rules out a number of algorithms that are specific to a particular learner (e.g., Caruana's multitask learning for neural networks [25]). The reason for this decision is that it allows a developer to easily apply any of these transfer algorithms to a problem for which a non-transfer solution has already been developed (such as the learning approaches described in Chapters 5 and 7, each of which worked best with a particular base learner).

A third category of algorithm described in Section 1.1 was using source data to tune parameters of a learning approach that would then be used to learn a model from target data. This method is implicitly used on all of the TAC data sets used in this chapter and the next, as the choices of base learning algorithms and the features of the data sets have been chosen to give good performance (in a non-transfer setting). More explicit methods of parameter tuning exist, such as algorithms that use source data to form priors over model parameters, but again this chapter does not consider such algorithms because they are specific to particular base learners (e.g., Gaussian processes [65]).

8.2.1 Base Learner on Target Data

The simplest possible baseline is to apply the base learner to the target data alone, ignoring all source data. Any transfer algorithm that performs worse than this baseline is said to exhibit *negative transfer*.

Algorithm 1 `Best Source Model`

Input a labeled data set $T = \{(x_i, y_i)\}$ and a set of source models $H^B = \{h^1, \ldots, h^B\}$.

Find the mean squared error e_b of each source model h^b on T.

Output the model h^b for which e_b is smallest.

8.2.2 Best Source Model

As another simple baseline, we can simply learn a model for each source data set using the base learner, and then choose the model that results in the lowest error on the target training data. This approach is shown as Algorithm 1.

8.2.3 Best Source Weighting

Another simple baseline is to combine the source and target data sets into one data set and apply the base learner to this full data set. However, if the target data set is much smaller than the source data sets, then the model learned may not be accurate on target data. To address this problem, we can reduce the weight given to source instances, assuming that our base learner can work with weighted instances. The challenge is to identify the correct weighting. We do this by decreasing the weight of each source instance from the initial weight (equal to the weight of each target instance) to zero in S steps and using n-fold cross validation to determine which of these weightings results in the lowest error on the target data. Note that all source weights are equal at each step–we are not attempting to determine optimal weights on each individual source instance. This approach is shown as Algorithm 2.

Algorithm 2 `Best Source Weighting`

Input two labeled data sets T_{source} and T_{target}, the number of steps S, the number of folds F for cross validation, and a base learning algorithm *Learner*.
For $t = 0, \ldots, S$:
1. Combine T_{source} and T_{target} into a single data set T. Assign a weight of 1 to each target instance and a weight of $1 - t/s$ to each source instance.
2. Call *Learner* with weighted data set T to obtain $model_t$.
3. Use F-fold cross validation to obtain an estimate $error_t$ of the error of $model_t$.

Output $model_t$ where $t = argmin_i\ error_i$.

8.2.4 Transfer Stacking

The method of using the best source model is restricted to using only one source model, meaning that the information contained in any additional source models is ignored. As an alternative, we could seek to combine the outputs of multiple source models. In fact, we could further extend this approach by also combining the outputs of these source models with the output of a model trained on the target data. I now introduce such an approach that we call *transfer stacking*, due to its similarity to stacking (or stacked generalization) [111].

Stacking is an ensemble approach in which a meta-level model combines multiple base models, all trained independently on the same set of data using different learning algorithms. The meta-level model is learned (typically using linear regression) from a meta-level data set created as follows. For each instance in the original training set, a meta-level instance is created using the outputs of each base model as features and using the original label. Cross validation is performed for each base learner so that the output for each instance in the original training set is obtained when it is out-of-sample. Once the meta-level model is learned, a new instance is handled by using the model to combine the outputs of the base models on the instance.

Here, instead of multiple base learners, we consider a single base learner and

Algorithm 3 `Transfer Stacking`

Input a labeled data set $T = \{(x_i, y_i)\}$ of size n, a set of source models $H^B = \{h^1, \ldots, h^B\}$, the number of folds F for cross validation, and a base learning algorithm *Learner*.

1. Let $O_{i,j} = h^j(x_i)$ for $1 \leq i \leq n$ and $1 \leq j \leq B$.
2. Perform F-fold cross validation on T using *Learner*. For $1 \leq i \leq n$, let $O_{i,B+1}$ equal the output of the learned model for the fold where instance i is in the validation set.
3. Call *Learner* with the full training set T and get model h^{B+1}.
4. Perform linear least squares regression on the system of equations $(\sum_{j=1}^{B+1} a_j O_{i,j}) + a_{B+2} = y_i$ for $1 \leq i \leq n$; that is, find the linear combination of models that minimizes squared error on T.

Output the model:
$h_f(x) = a_1 h^1(x) + ... + a_{B+1} h^{B+1}(x) + a_{B+2}$

(potentially) multiple source models previously trained on source data; hence when building the meta-level data set, cross validation is required only for determining the outputs of the base learner on the target data set, not the source models. The full details of transfer stacking are shown in Algorithm 3. Although past work on using stacking for regression [20] has shown that placing constraints on the coefficients of the linear meta-level model (such as requiring them to be positive or to sum to one) can be helpful, we did not find it to improve results in our experiments, and so we omit this detail.

8.2.5 Feature Augmentation

The next two transfer methods depend on augmenting the feature space of the data. The first method, which I will refer to as *model feature augmentation*, simply trains a model on each source data set, and then adds the output of each model as an additional feature of the target data. As a result, the learner should be able to identify those situations in which the source models are relevant dur-

Algorithm 4 `Model Feature Augmentation`

Input a labeled data set $T = \{(x_i, y_i)\}$ of size n, a set of source models $H^B = \{h^1, \ldots, h^B\}$, and a base learning algorithm *Learner*.
For $i = 1, \ldots, n$:

1. Let o_i be the vector $\langle h^1(x_i), \ldots, h^B(x_i)\rangle$.
2. Let $x_i' = \langle x_i : o_i\rangle$.

Let $T' = \{(x_i', y_i)\}$, and call *Learner* with T' to get model m.
Output the model m.

ing training and then make use of their outputs appropriately when labeling new instances. This method has been tested on classification problems [33] and found to perform reasonably well, and no changes are required to apply this method to regression problems. Details are given in Algorithm 4.

The second feature augmentation method, which I will refer to as *source feature augmentation*, involves training the learner on a combination of the target data and all source data sets. This method was introduced in [32], where it was shown to outperform several other methods on classification tasks. Again, no changes are required to apply this method to regression tasks. As in model feature augmentation, each instance is augmented with new features; however, this time the features that are added depend on which data set the instance came from. 1+B new copies of the feature space are added, corresponding to the 1+B data sets. (Recall that B is the number of source data sets.) For each instance, each of these copies is left blank unless it corresponds to the data set from which the instance originally came, in which case all values are copied exactly from the original features. To illustrate, consider the situation in which we have one source data set (B=1), one target instance $x_t \in X$, and one source instance $x_s \in X$. Each augmented instance will have the form $\langle a, b, c\rangle$, where a, b, and c are vectors in X. Instance x_t will become $\langle x_t, x_t, \mathbf{0}\rangle$, and instance x_s will become $\langle x_s, \mathbf{0}, x_s\rangle$, where $\mathbf{0} \in X$ is the zero vector. Full details are given in Algorithm 5.

Algorithm 5 `Source Feature Augmentation`

Input one labeled target data set T_0 and B labeled source data sets $T_1, \ldots, T_B$, all of form $\{(x_i^b, y_i^b)\}$, and a base learning algorithm *Learner*.
For $b = 0, \ldots, B$:
1. **For** $i = 1, \ldots, |T_b|$:
   Let $\hat{x}_i^b$ be the vector $\langle x_i^b : f_0(x_i^b) : \ldots : f_B(x_i^b) \rangle$, where $f_k(x_i^b) = x_i^b$ if $k = b$ and **0** otherwise.
2. Let $T'_b = \{(\hat{x}_i^b, y_i^b)\}$.

Let $T = T'_0 \cup \ldots \cup T'_B$, and call *Learner* with T to get model m.
Output the model m.

The idea behind this method is that for a feature that has the same meaning across data sets, the learner will make use of the "original" copy of the feature that is shared across all data sets, effectively transferring knowledge about the meaning of that feature. For a feature that has different effects on different data sets, the learner should only make use of the data-set-specific copies of that feature, preventing negative transfer.

8.2.6 Boosting Approaches

One approach that has been applied successfully to classification transfer problems is boosting. Boosting is an ensemble method in which a sequence of models (or hypotheses) $h_1 \ldots h_N$, each mapping from X to Y, are iteratively fit to some transformation of a data set using a base learner. The outputs of these models are then combined into a final hypothesis h_f. In this section, I describe two existing boosting-based algorithms for classification transfer, and then discuss extensions of these algorithms that are necessary for our regression setting. Again, each algorithm fits into one of the two categories discussed previously: algorithms that make use of models trained on the source data, and algorithms that use the source data directly as training data.

In ExpBoost [78], a separate hypothesis (or expert, hence the name) h^i is learned for each of the B source data sets, and learning is performed using only T_{target}. At each step of the boosting process, ExpBoost chooses to use either the hypothesis h_t learned from the weighted training data or one of the experts, depending on which is most accurate.

In contrast, TrAdaBoost [31] uses the source data sets directly by combining them with T_{target} to form a single data set. At each boosting step, TrAdaBoost increases the relative weights of target instances that are misclassified. When a source instance is misclassified, however, its weight is decreased. In this way, TrAdaBoost aims to identify and make use of those source instances that are most similar to the target data while ignoring those that are dissimilar.

I provide additional details on these algorithms and their extensions below, but first I address the issue of applying boosting algorithms to regression problems.

8.2.6.1 AdaBoost and Regression

One of the best known boosting methods for classification, and the one upon which ExpBoost and TrAdaBoost are based, is AdaBoost (specifically, AdaBoost.M1) [40]. In AdaBoost, each training instance receives a weight w_i that is used when learning each hypothesis; this weight indicates the relative importance of each instance and is used in computing the error of a hypothesis on the data set. After each iteration, instances are reweighted, with those instances that are not correctly classified by the last hypothesis receiving larger weights (as in step 5 of Algorithm 6). Thus, as the process continues, learning focuses on those instances that are most difficult to classify.

A number of methods have been proposed for modifying AdaBoost for regression, and as TrAdaBoost and ExpBoost are based on AdaBoost, these modifications can be used on them as well. In our work, we explored two of these methods that have been shown to be generally effective and that can be applied to TrAdaBoost and ExpBoost in a straightforward way: AdaBoost.R2 and AdaBoost.RT.

The key to AdaBoost is the reweighting of those instances that are misclassified at each iteration. In regression problems, the output given by a hypothesis h_t for an instance x_i is not correct or incorrect, but has a real-valued error $e_i = |y_i - h_t(x_i)|$ that may be arbitrarily large. Thus, we need a method of mapping an error e_i into an adjusted error e'_i that can be used in the reweighting formula used by AdaBoost.

The method used in AdaBoost.R2 [37] is to express each error in relation to the largest error $D = max_{i=0}^{n}|e_i|$ in such a way that each adjusted error e'_i is in the range $[0, 1]$. In particular, one of three possible loss functions is used: $e'_i = e_i/D$ (linear), $e'_i = {e_i}^2/D^2$ (square), or $e'_i = 1 - exp(-e_i/D)$ (exponential). The degree to which instance x_i is reweighted in iteration t thus depends on how large the error of h_t is on x_i relative to the error on the worst instance. AdaBoost.RT [93], on the other hand, continues to label each output as correct ($e'_i = 0$) or incorrect ($e'_i = 1$) using an error threshold ϕ. That is, if $e_i > \phi$, then $e'_i = 1$; otherwise, $e'_i = 0$. This method has the advantage that, unlike AdaBoost.R2, extreme outliers do not have the potential to skew the reweighting and receive relatively large weights; however, the threshold ϕ must be set properly.

In preliminary experiments, we found AdaBoost.R2 with the linear loss function to work consistently well, and were unable to find values of ϕ that allowed AdaBoost.RT to regularly match this performance. (All preliminary experiments were conducted using the data sets and methodology described in Sections 8.4.1 and 8.4.2.) Henceforth, I discuss only AdaBoost.R2 with the linear loss function, shown in Algorithm 6.

It is worth noting that AdaBoost.R2 will terminate before the specified number of iterations if the hypothesis learned by any iteration has an adjusted

Algorithm 6 `AdaBoost.R2` [37]

Input the labeled target data set $T = \{(x_i, y_i)\}$ of size n, the maximum number of iterations N, and a base learning algorithm *Learner*. Unless otherwise specified, set the initial weight vector $\mathbf{w}^1$ such that $w_i^1 = 1/n$ for $1 \leq i \leq n$.
For $t = 1, \ldots, N$:
1. Call *Learner* with the training set T and the distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
2. Calculate the adjusted error e_i^t for each instance:

   let $D_t = max_{j=1}^{n} |y_j - h_t(x_j)|$

   then $e_i^t = |y_i - h_t(x_i)|/D_t$
3. Calculate the adjusted error of h_t:

   $\epsilon_t = \sum_{i=1}^{n} e_i^t w_i^t$; if $\epsilon_t \geq 0.5$, stop and set $N = t - 1$.
4. Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
5. Update the weight vector:

   $w_i^{t+1} = w_i^t \beta_t^{1-e_i^t}/Z_t$ (Z_t is a normalizing constant)

Output the model:
$h_f(x) =$ the weighted median of $h_t(x)$ for $1 \leq t \leq N$, using $\ln(1/\beta_t)$ as the weight for hypothesis h_t.

error of 0.5 or above. This stopping condition is left unchanged from AdaBoost. In a classification setting, an error above 0.5 means that the hypothesis is wrong more often than it is right, and stopping if this occurs is reasonable (and also necessary to preserve certain theoretical properties). In our regression setting using the linear loss function of AdaBoost.R2, an error above 0.5 means that the weighted average of errors is more than half of the largest error. Whether this is undesirable or not likely depends on the nature of the concept being learned. In our preliminary experiments, we found that removing this stopping condition (and setting an upper bound of 0.99 on β_t) decreased performance somewhat, so it may be that this stopping condition guards against overfitting. Identifying a more suitable stopping condition (or experimenting with cross validation) is an area for future work.

8.2.6.2 ExpBoost.R2

Combining the principles of AdaBoost.R2 with those of ExpBoost results in the new regression algorithm ExpBoost.R2. In addition to the target data, ExpBoost.R2 receives as input a set of source experts $H^B = \{h^1, \ldots, h^B\}$. The steps involving computing the adjusted error and outputting the final model correspond to the same steps from Algorithm 6. The primary difference is in step 1 of each boosting iteration. After obtaining h_t, ExpBoost.R2 computes the weighted errors of each expert in H^B on the current weighting of T, and if any expert has a lower weighted error than h_t, h_t is replaced with the best expert. ExpBoost.R2 is shown as Algorithm 7.

In step 2 of ExpBoost.R2, we use cross validation to calculate the error of h_t, the hypothesis for the current iteration. Cross validation is not used in the original ExpBoost algorithm, but we found it to be necessary in our preliminary experiments. We observed that h_t was typically able to fit the data it was trained on well, and thus it was usually chosen as h_t^{best} over any of the experts in step 3, preventing effective transfer. When cross validation was used to evaluate h_t on out-of-sample data, experts were chosen as h_t^{best} much more frequently, and the performance of the algorithm improved. We found $F = 5$ to be sufficient in our experiments.

8.2.6.3 TrAdaBoost.R2

Combining the principles of AdaBoost.R2 with those of TrAdaBoost results in the new regression algorithm TrAdaBoost.R2. TrAdaBoost.R2 takes two data sets as input, T_{target} and T_{source}, of size n and m, respectively, and combines them into a single set T used in boosting. Although the original work on TrAdaBoost does not consider the issue of multiple sources, we are interested in cases where any number of sources may exist. When there is more than one source, we simply

Algorithm 7 `ExpBoost.R2`

Input the labeled target data set $T = \{(x_i, y_i)\}$ of size n, a set of experts $H^B = \{h^1, \ldots, h^B\}$, the maximum number of iterations N, the number of folds F for cross validation, and a base learning algorithm *Learner*. Set the initial weight vector $\mathbf{w}^1$ such that $w_i^1 = 1/n$ for $1 \leq i \leq n$.

For $t = 1, \ldots, N$:

1. Call *Learner* with the training set T and the distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
2. Calculate the error of each hypothesis $h^i \epsilon H^B$ as $\sum_{j=1}^{n} w_j^t |y_j - h^i(x_j)|$. For h_t, calculate the error similarly, except that F-fold cross validation should be used with *Learner* to obtain an error estimate instead of directly computing the error of h_t.
3. Let h_t^{best} be the hypothesis in $H^B \cup h_t$ with the lowest error.
4. Calculate the adjusted error e_i^t for each instance:

   let $D_t = max_{j=1}^{n} |y_j - h_t^{best}(x_j)|$

   then $e_i^t = |y_i - h_t^{best}(x_i)|/D_t$
5. Calculate the adjusted error of h_t^{best}:

   $\epsilon_t = \sum_{i=1}^{n} e_i^t w_i^t$

   If $\epsilon_t \geq 0.5$, stop and set $N = t - 1$.
6. Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$.
7. Update the weight vector:

   $w_i^{t+1} = w_i^t \beta_t^{1-e_i^t}/Z_t$ (Z_t is a normalizing constant)

Output the model:
$h_f(x) =$ the weighted median of $h_t^{best}(x)$ for $1 \leq t \leq N$, using $\ln(1/\beta_t)$ as the weight for hypothesis h_t^{best}.

combine all source data sets into a single data set. As TrAdaBoost.R2 handles the reweighting of each training instance separately, there should be no harm in mixing data in this fashion, but care should be taken in setting the initial weight vector. Our experiments involve source data sets of (roughly) equal sizes, and so we simply assign all source instances (and target instances) the same initial weight. If one source data set were larger than another, however, setting weights uniformly would result in more emphasis being given to that source, at least in early boosting iterations.

As with ExpBoost.R2, the steps involving computing the adjusted error correspond to the same steps from Algorithm 6. The primary difference between TrAdaBoost.R2 in Algorithm 8 and AdaBoost.R2 in Algorithm 6 is in step 5 of each iteration. Instead of treating all data equally, TrAdaBoost.R2 increases the weights of target instances by setting $w_i^{t+1} = w_i^t \beta_t^{-e_i^t}/Z_t$ and decreases the weights of source instances by setting $w_i^{t+1} = w_i^t \beta^{e_i^t}/Z_t$, where $\beta = 1/(1 + \sqrt{2 \ln n/N})$. In addition, TrAdaBoost.R2 considers only the final $\lceil N/2 \rceil$ hypotheses when taking the weighted median to determine output (as a result of theoretical considerations in the original TrAdaBoost). TrAdaBoost.R2 is shown as Algorithm 8.

8.2.6.4 Two-stage TrAdaBoost.R2

In analyzing the performance of TrAdaBoost.R2, we observed it to be highly susceptible to overfitting (that is, beyond some point, accuracy decreased as the number of boosting iterations N increased). In contrast, AdaBoost.R2 and ExpBoost.R2 do not appear to suffer from this problem. After experimenting with cross validation to select N, we still saw mixed performance from TrAdaBoost.R2. Closer inspection of the results revealed two problems. First, when the size of T_{source} is much larger than T_{target}, it can take many iterations for the total weight of the target instances to approach the total weight of the source instances, and by this time the weights of the target data may be heavily skewed–those target instances that are either outliers or most dissimilar to the source data may represent most of the weight. Second, even those source instances that are representative of the target concept tend to have their weights reduced to zero eventually. The use of the adjusted error scheme from AdaBoost.R2 is the reason. Whereas in TrAdaBoost the relevant source instances will generally be classified correctly and not have their weights reduced, in TrAdaBoost.R2 even small errors lead to weight re-

Algorithm 8 `TrAdaBoost.R2`

Input two labeled data sets T_{source} (of size n) and T_{target} (of size m), the maximum number of iterations N, the initial weight vector (of total weight 1) $\mathbf{w}^1 = (w_1^1, \ldots, w_{n+m}^1)$, and a base learning algorithm *Learner*. Let T be the combination of T_{source} and T_{target} such that the first n instances in T are those from T_{source}.

For $t = 1, \ldots, N$:

1. Call *Learner* with the combined training set T and the distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
2. Calculate the adjusted error e_i^t for each instance:

   let $D_t = max_{j=n+1}^{n+m} |y_j - h_t(x_j)|$

   then $e_i^t = |y_i - h_t(x_i)|/D_t$
3. Calculate the adjusted error of h_t on T_{target}:

   $\epsilon_t = \sum_{i=n+1}^{n+m} e_i^t w_i^t / \sum_{j=n+1}^{n+m} w_j^t$

   If $\epsilon_t \geq 0.5$, stop and set $N = t - 1$.
4. Let $\beta_t = \epsilon_t/(1 - \epsilon_t)$ and $\beta = 1/(1 + \sqrt{2 \ln n/N})$.
5. Update the weight vector:

   $$w_i^{t+1} = \begin{cases} w_i^t \beta^{e_i^t}/Z_t, & 1 \leq i \leq n \\ w_i^t \beta_t^{-e_i^t}/Z_t, & n+1 \leq i \leq n+m \end{cases}$$

   where Z_t is a normalizing constant.

Output the model:
$h_f(x) =$ the weighted median of $h_t(x)$ for $\lceil N/2 \rceil \leq t \leq N$, using $\ln(1/\beta_t)$ as the weight for hypothesis h_t.

ductions. The fact that TrAdaBoost.R2 uses only the hypotheses generated during the final half of boosting iterations exacerbates this problem. (We also tried using all hypotheses, with mixed results.)

To address these problems, we designed a version of TrAdaBoost.R2 that adjusts instance weights in two stages. In stage one, the weights of source instances are adjusted downwards gradually until reaching a certain point (determined through cross validation). In stage two, the weights of all source instances are frozen while the weights of target instances are updated as normal in AdaBoost.R2. Only the hypotheses generated in stage two are stored and used to determine the output of the resulting model. We call this algorithm two-stage TrAdaBoost.R2, and it is shown as Algorithm 9. Note that if the ideal stopping point for reducing source weights were known in advance, stage 1 would conclude before stage 2 began. However, because we are using cross validation to determine when to stop reducing source weights, in Algorithm 9 stage 2 (line 1) is embedded inside stage 1 (the remainder of the for-loop). Also, the weighting factor β_t is not chosen based on the hypothesis error, as before, but is chosen to result in a certain total weight for the target instances. In this way, the total weight of the target instances increases uniformly from $m/(n+m)$ to 1 in S steps. In our implementation, we approximated the value of β_t satisfying the conditions shown in Algorithm 9 using a binary search. In addition, it is not necessary to progress through all S steps once it has been determined that errors are increasing.

8.2.7 TrBagging

In addition to boosting methods for classification transfer, there also exists a bagging method, TrBagg, that has been shown to outperform other classification transfer methods (including TrAdaBoost and source feature augmentation) on a number of collaborative tagging tasks [51]. TrBagg can be applied to regression problems without any significant modifications, and is shown as Algorithm 10.

The original bagging method [19] (bagging stands for "bootstrap aggregating") is as follows. Given a base learner, a number of iterations N, and a training set T, bagging uses bootstrap sampling (sampling with replacement) on T to create N data sets of size $|T|$, and then learns a model using the base learner on each set. To classify a new instance, the label output by the majority of the N models is used. Bagging can lead to significantly lower error than using the base learner alone, especially when the base learner is unstable (i.e., slight changes in

Algorithm 9 `Two-stage TrAdaBoost.R2`

Input two labeled data sets T_{source} (of size n) and T_{target} (of size m), the number of steps S, the maximum number of boosting iterations N, the number of folds F for cross validation, and a base learning algorithm *Learner*. Let T be the combination of T_{source} and T_{target} such that the first n instances in T are those from T_{source}. Set the initial weight vector $\mathbf{w}^1$ such that $w_i^1 = 1/(n+m)$ for $1 \le i \le n+m$.
For $t = 1, \ldots, S$:

1. Call AdaBoost.R2′ with T, distribution $\mathbf{w}^t$, N, and *Learner* to obtain $model_t$, where AdaBoost.R2′ is identical to AdaBoost.R2 except that the weights of the first n instances are never modified. Similarly, use F-fold cross validation to obtain an estimate $error_t$ of the error of $model_t$.
2. Call *Learner* with T and distribution $\mathbf{w}^t$, and get a hypothesis $h_t : X \to \mathbb{R}$.
3. Calculate the adjusted error e_i^t for each instance as in AdaBoost.R2.
4. Update the weight vector:

$$w_i^{t+1} = \begin{cases} w_i^t \beta_t^{e_i^t}/Z_t, & 1 \le i \le n \\ w_i^t/Z_t, & n+1 \le i \le n+m \end{cases}$$

where Z_t is a normalizing constant, and β_t is chosen such that the resulting weight of the target (final m) instances is $\frac{m}{(n+m)} + \frac{t}{(S-1)}(1 - \frac{m}{(n+m)})$.

Output $model_t$ where $t = argmin_i \; error_i$.

---

the training data greatly change the model it learns).

TrBagg also learns models on several different sampled data sets. The target and all source data sets are combined into one data set, and N subsets of size $|T_{target}|$ are sampled (with replacement) from this set. A model is trained using the base learner on each subset, as shown in the learning phase of Algorithm 10. Unlike bagging, however, TrBagg does not make use of all of these models. While some subsets may contain many instances that come from the target set or are relevant source instances, other subsets may contain many irrelevant source instances. To choose which models to use, TrBagg performs a filtering phase, also shown in Algorithm 10. Ideally, we would be able to find the optimal subset of models to use, but this is not feasible. Instead, TrBagg evaluates each model on the target set and sorts the models in order of increasing error. TrBagg then evaluates subsets of the M most accurate models, for $0 < M \leq N$, and chooses the subset with the lowest error on the target data. To prevent negative transfer, however, TrBagg includes a *fallback model* trained on only the target data in each subset; if no additional models prove useful, then the lowest error will occur when $M = 0$, and the fallback model will be used alone. To prevent TrBagg from always finding $M = 0$ to be optimal, it is important that the fallback model be "trained so as not to over-fit to target training data" [51]. In our implementation of TrBagg, we ensure that this is the case by using linear regression. As this is a weaker learning algorithm than the base learners we will consider, we would expect it to be less accurate than many of the models trained by TrBagg on subsets, and this was nearly always the case in our experiments. For use with regression problems, we take the median of the model outputs instead of the majority output.

8.2.8 Discussion of Algorithms

Table 8.1 summarizes the properties of the nine regression transfer algorithms described. As has been mentioned, the transfer algorithms presented can be classified as either using source models or using the source data directly. Each approach has its advantages. Using the source data directly provides an algorithm with information that is lost when only models are used. If instances from a particular source data set cover only a certain region of the input space X, then it is likely that that source will only influence the output of the resulting model for inputs from that region. Similarly, if only a portion of a source data set is relevant to the target concept, then it may be possible for an algorithm given the source data to identify the relevant instances. On the other hand, if a particular source concept is especially similar or dissimilar to the target concept, an algorithm using source experts can easily learn to make heavy or light use of that source's model,

Algorithm 10 `TrBagg [51]`

Input two labeled data sets T_{source} (which may be the combination of many source data sets) and T_{target}, the number of iterations N, and a base learning algorithm *Learner*. Let T be the combination of T_{source} and T_{target}.

Learning phase:
For $n = 1, \ldots, N$:
1. Sample (with replacement) a data set T_n of size $|T_{target}|$ from T.
2. Train model h_n on T_n using the base learner.
3. Evaluate the error of h_n on T_{target}.

Sort the models in order of increasing error (so h_1 is now the model with the lowest error).

Filtering phase:
Train fallback model h_0 on T_{target} using linear regression.
Let $S_{best} = h_0$.
Let e_{best} = the error of h_0 on T_{target}.
For $n = 0, \ldots, N$:
1. Let $S = \{h_0, \ldots, h_n\}$.
2. Let e be the error of taking the median output of S on T_{target}.
3. If $e < e_{best}$, then $e_{best} = e$ and $S_{best} = S$.

Output the model:
$h_f(x)$ = the median output of each model in S_{best} on x

while the algorithms described here that use source data have no way of knowing which source set each source instance comes from (although it may be possible to design an algorithm that makes use of such information). Perhaps the biggest advantage of methods that use source models is running time. In a situation in which source models can be trained in advance and the amount of source data greatly exceeds the amount of target data available, learning using source models may be many times faster than learning from source data directly. For instance, for the largest data sets considered in this chapter (the TAC data sets described in Section 8.4.3), running times of algorithms using source models were often under a minute, while running times of algorithms using source data were often several

Algorithm	*Uses source data*	*Uses source model*	*Modifies data*	*Ensemble method*	*Uses Boosting*	*Uses cross validation*
best source model		✓		✓		
best source weighting	✓					
transfer stacking		✓		✓		✓
model feat. augment		✓	✓			
source feat. augment	✓		✓			
ExpBoost.R2		✓		✓	✓	✓
TrAdaBoost.R2	✓			✓	✓	✓
2-St. TrAdaBoost.R2	✓			✓	✓	✓
TrBagg	✓			✓		

Table 8.1: Properties of the regression transfer algorithms

minutes, even when only a handful of target data was available.

Of the algorithms using source models, ExpBoost.R2 and transfer stacking are most similar in that both generate a weighting of source models and models trained on the target data. One possible disadvantage of ExpBoost.R2 is that at each boosting iteration, it must choose between either the newly learned hypothesis or a single source model. As a result, ExpBoost.R2 might be expected to consistently choose a particular source model if it is fairly accurate, or to ignore the source models altogether if they are fairly inaccurate. In fact, we observed this behavior in a number of experiments. Transfer stacking, on the other hand, is not so restricted, and as a result it generally makes heavier use of various source models and performs somewhat better in our experiments. Model feature augmentation is different from ExpBoost.R2 and transfer stacking in that it is capable of learning to treat source models differently at different parts of the input space. As an example, if using M5P model trees (which perform linear regression at each leaf) as the base learner, then the output of both transfer stacking and model feature augmentation will be a linear combination of the (non-augmented) input features and source model outputs. However, the coefficients of the source model outputs will be chosen at the leaf level in model feature augmentation (as they are treated as features), while the coefficients will be chosen at the global level in transfer stacking (after the tree is built). (In fact, we also experimented with a modified version of M5P model trees in which transfer stacking was performed at the leaves, but results were mixed.) If a source concept is similar to the target concept only in certain regions of the input space, then model feature augmentation might be expected to outperform ExpBoost.R2 and transfer stacking.

As mentioned above, one advantage of using source data directly is that an algorithm can identify which source instances are most relevant to the target concept. Both (two-stage) TrAdaBoost.R2 and TrBagg attempt to do this, though in different ways. TrAdaBoost.R2 directly evaluates the error on each source instance at each iteration and reduces source weights accordingly. TrBagg identifies irrelevant source instances in a more indirect way by ranking its models according to their error on the target set. Presumably, if an irrelevant source instance is sampled in a given bagging iteration, then the model learned at that iteration will have a higher error and be less likely to be included in the final result. Source feature augmentation is not capable of ignoring less relevant source instances; however, if these instances occur within a particular region of the input space for a certain source set, then source feature augmentation may be able to identify these instances (using the added features specific to that source) and learn to handle them differently than other instances.

One final point that must be noted is that boosting alone (and to a lesser degree, bagging) is capable of greatly reducing the errors of the models learned. In fact, in nearly all of the experiments described below, using boosting lead to a statistically significant decrease in error when training on target data alone. As a result, the transfer algorithms that make use of boosting have an inherent advantage over the algorithms that do not, even if the use of boosting does not actually aid transfer. To allow a fair comparison between algorithms, we apply boosting (using AdaBoost.R2) to all algorithms that do not already perform boosting. For those ensemble methods which themselves require a base learner (transfer stacking and TrBagg), we face the question of whether boosting should be applied at the level of the base learner, or at the top level (after the algorithm has built a complete model). For instance, with transfer stacking, we could boost the base learner and then form a single linear combination of the result with the source models, or we could boost the entire transfer stacking process, resulting in a unique linear combination of source models and (unboosted) learned models at each boosting iteration. We found the second approach to result in more consistent performance in our preliminary experiments, and so that is the approach we use. The addition of boosting generally had a positive impact on the performance of those algorithms that did not previously use it.

8.3 Data Transformation

In the formulation of our regression transfer problem in Section 8.1, we assumed that both source and target concepts had labels in the same output space. In many regression settings in which we might consider transfer, however, different concepts might have labels with considerably different label distributions. While we largely view this as a data preparation issue (e.g., labels can be expressed in comparable terms, such as using relative instead of absolute prices in financial data) and thus beyond the scope of the problem studied here, in our experiments we do take some simple measures to ensure similar label distributions.

In algorithms making use of models trained on source data, we can directly modify the models so that their outputs on the target data fall in an appropriate range. We do so by evaluating the models on the target training data (thus making use only of data available to the learner) and performing linear regression to find the linear transformation that best fits the outputs to the true labels. This transformation is then applied whenever the model is used by the learning algorithm. In algorithms using the source data directly, for each source data set we train a model on the set, find the linear transformation in the same manner, and then apply this transformation to the labels in the source data set before passing it to the learning algorithm. In general, for the data sets described in the following section, we found that this procedure was worthwhile, as it often resulted in a significant increase in accuracy while only occasionally producing a slight decrease in accuracy. (Trying regression with higher degree polynomials tended to produce modest improvements at best and large decreases in accuracy at worst.) The TAC data sets of Section 8.4.3 were the exception to this observation, as no significant improvement was seen from transforming the data. This result is consistent with our assertion that the issue here is one of data preparation–the TAC labels are already expressed in formats (relative prices or probabilities) that have similar distributions across different concepts. We therefore omit the data transformation step on experiments involving TAC data sets.

8.4 Experiments

We evaluated the transfer algorithms described in Section 8.2 on nine different problems: four data sets from the UCI Repository, a space of artificial data sets created from a known function, and four prediction problems from TAC. Experiments were performed using the WEKA 3.4 [110] machine learning package with default parameters for the base learners. In the first group of experiments, we tested two base learners. For the remainder, we used the regression algorithm in WEKA giving the lowest error when used alone as the base learner. The number of boosting iterations for each algorithm was set to 30, as this number appeared sufficient to reach a plateau without overfitting for each algorithm and data set. (The one exception was TrAdaBoost.R2, for which we used cross validation to set the number of boosting iterations to obtain the best possible result.) For two-stage TrAdaBoost.R2 and best source ratio, we also set S (the number of steps for trying different source weightings) to 30. Ten folds were used for cross validation where required. For TrBagg, we used 100 bagging iterations. Source models were generated by running AdaBoost.R2 on a complete source data set with the appropriate base learner. We used AdaBoost.R2 as the baseline non-transfer algorithm in each experiment as it consistently produced lower errors than using the base learner alone and offers a fair comparison against boosting transfer algorithms. For all experiments, each result represents the average RMS error over 30 runs. Results said to be significant are statistically significant ($p < .05$) according to paired t-tests.

8.4.1 Four UCI Data Sets

The first set of experiments evaluates all ten algorithms described above on four data sets taken from the UCI Machine Learning Repository:[1] concrete strength, housing, auto MPG, and automobile. (We chose the first four data sets that represented standard regression problems and had a few hundred instances; no other data sets were tried.) We divide these standard regression data sets into target and source sets by using a variation on the technique used by Dai et al. [31] in a classification setting. For each data set, we identify a continuous feature that has a moderate degree of correlation (around 0.4) with the label. We then sort the instances by this feature, divide the set in thirds (low, medium, and high), and remove this feature from the resulting sets. By dividing based on a feature moderately correlated with the label, we hope to produce three data sets that represent slightly different concepts; if the correlation were zero, the concepts might be identical, and if the correlation were high, the concepts might be significantly different and have very different label ranges. In each experiment, we use one data set as the target and the other two as sources, for a total of 12 experiments.

Table 8.2 shows the results of all ten learning algorithms on all 12 experiments using both M5P model trees and neural networks as base learners. Target data training sets contained 25 instances. (Increasing this number resulted in

[1] http://archive.ics.uci.edu/ml/index.html

qualitatively similar results.) Source data sets ranged from 68 to 343 instances. Numbers in bold represent results that are among the best–either the lowest error, or not significantly higher. Numbers in italics represent results that are not significantly better than AdaBoost.R2 (which uses target data alone), that is, those where transfer failed. All other numbers represent results that were not among the best, but where there was significant transfer. Table 8.3 summarizes these results by listing the number of times each algorithm fell into each performance category, and lists algorithms in increasing order of outperforming AdaBoost.R2.

The best source model is significantly better than AdaBoost.R2 exactly half of the time, but is sometimes much worse, suggesting that the degree of similarity between source and target data sets varies considerably across the range of experiments. Not surprisingly, the cases where the best source model fares worst are often those where other algorithms that use source models fare poorly.

ExpBoost.R2 performs poorly, beating AdaBoost.R2 significantly only six out of 24 times. Transfer stacking does much better, beating AdaBoost.R2 significantly 15 times, suggesting that there is a benefit to considering linear combinations of models at each boosting step instead of only individual models.

The algorithms that perform feature augmentation had fairly mediocre performance. Comparing model feature augmentation to transfer stacking, model feature augmentation had a lower error only nine out of 24 times, despite the aforementioned advantage of being able to treat source models differently for different parts of the input space.

TrAdaBoost.R2 (with the number of boosting iterations chosen using cross validation) gives promising but somewhat erratic results, beating AdaBoost.R2 significantly 16 times but performing much worse in a few cases. Two-stage TrAdaBoost.R2 produces much better results and is the clear winner in this set of experiments, finishing among the top algorithms 21 out of 24 times and never failing to significantly beat AdaBoost.R2. TrBagg is also fairly successful, usually outperforming AdaBoost.R2 but failing to be among the best algorithms. Interestingly, simply finding the best source weighting also performs well, significantly outperforming AdaBoost.R2 17 times.

Overall, these results suggest that making use of source data directly is more effective than using source models. We hypothesize that the reason is that algorithms that use source data are better able to make use of only relevant source instances, as discussed above. It is notable that Two-stage TrAdaBoost.R2 and TrBagg, the two algorithms that directly address the issue of instance relevance, are the most consistently successful in realizing a gain from transfer. Remaining experiments (particularly those of Sections 9.1 and 9.4) will confirm that Two-stage TrAdaBoost.R2 is particularly effective at distinguishing between relevant and irrelevant source instances. On the other hand, source feature augmentation attempts to identify relevant features rather than relevant instances and is the least effective of the algorithms making use of source data. Investigation of the models output by source feature augmentation shows that it tends to make heavy use of the source-specific features, thus limiting the potential for transfer.

While less successful overall, algorithms that use source models still had the best performance in a few cases, and as noted previously, they are generally much less computationally intensive, due to using smaller amounts of data (target data only).

For the remaining experiments, I omit the results of the best source model, best source weighting, TrAdaBoost.R2, and ExpBoost.R2, as these algorithms continue to be outperformed by other algorithms based on similar principles.

Base Lrnr.	*Algorithm*	*Data set (divided into 3 subsets)*											
		Concrete Strength			Housing			Auto MPG			Automobile		
M5P	AdaBoost.R2 (target)	10.26	11.01	13.26	3.65	3.59	6.52	2.90	2.92	4.35	1963	3576	4893
	best source model	8.63	8.08	9.55	**2.98**	*5.27*	*9.98*	2.38	2.57	*4.44*	**1374**	*3741*	*6059*
	best source weighting	*10.25*	6.98	**8.66**	**2.99**	*3.42*	*6.52*	2.35	2.59	*4.33*	*1734*	2678	**2940**
	transfer stacking	8.47	7.48	10.03	**3.03**	*3.99*	*7.24*	2.30	2.75	*4.48*	**1325**	*3480*	*4811*
	model feat. augment	8.28	7.81	10.03	3.24	*3.52*	**6.28**	2.38	*2.92*	*4.31*	1717	*3483*	*4924*
	source feat. augment	**8.08**	7.48	9.44	3.37	*3.53*	6.36	2.28	2.72	*4.43*	*2225*	2702	3479
	ExpBoost.R2	*10.11*	9.64	11.76	**3.02**	*3.74*	*6.66*	2.53	*2.94*	*4.39*	*1791*	*3661*	*4932*
	TrAdaBoost.R2	*10.76*	7.04	9.71	3.38	*3.57*	*7.03*	2.19	2.58	**4.24**	*1815*	2851	3527
	2-St. TrAdaBoost.R2	**8.211**	**6.49**	**8.66**	**2.99**	**3.12**	**6.12**	**2.14**	**2.52**	**4.21**	1564	**2555**	3202
	TrBagg	9.30	9.46	11.24	3.38	3.30	*7.61*	2.34	*2.88*	**4.15**	1604	2955	3764
NN	AdaBoost.R2 (target)	10.47	11.95	14.84	3.89	3.67	7.54	2.76	3.55	5.17	1593	3200	3836
	best source model	*10.12*	9.67	13.87	*7.00*	*4.99*	*9.22*	*2.66*	2.77	4.43	*1481*	2484	*6119*
	best source weighting	*10.48*	**8.02**	10.77	*3.89*	**3.00**	**6.46**	2.44	2.80	**4.19**	1312	**2277**	**2858**
	transfer stacking	**9.49**	9.80	12.93	*3.75*	*3.58*	*7.74*	2.48	3.00	4.40	**1215**	2640	*3761*
	model feat. augment	10.17	9.31	11.98	*3.82*	*3.48*	*7.39*	*2.92*	3.27	4.86	1462	3063	3638
	source feat. augment	*13.12*	11.37	*17.05*	*3.91*	*4.24*	7.14	*2.80*	3.33	4.82	*2022*	2824	*4000*
	ExpBoost.R2	*10.14*	*11.62*	13.37	*3.88*	*3.66*	*7.63*	*2.79*	*3.50*	*5.20*	1392	*3174*	*3829*
	TrAdaBoost.R2	*11.34*	9.05	11.91	*4.02*	3.29	*7.68*	2.33	2.80	4.37	*1718*	2573	3268
	2-St. TrAdaBoost.R2	**9.69**	**8.09**	**9.92**	**3.27**	**2.99**	**6.45**	**2.14**	**2.60**	**4.18**	1290	**2276**	**2843**
	TrBagg	**9.67**	**8.33**	10.98	3.55	**2.95**	6.96	2.22	2.78	**4.17**	1391	2904	3651

Table 8.2: RMS error on four UCI datasets, each divided into three concepts, using M5P model trees and neural networks as base learners. **Bold**: lowest error; *Italic*: not significantly better than AdaBoost.R2 (95% confidence in each case).

Algorithm	*Better than target alone*	*Among best*
ExpBoost.R2	6	1
best source model	12	2
source feat. augment	14	1
model feat. augment	15	1
transfer stacking	15	4
TrAdaBoost.R2	16	1
best source weighting	17	9
TrBagg	22	5
2-St. TrAdaBoost.R2	24	21

Table 8.3: Summary of results in Table 8.2

8.4.2 Friedman #1

Friedman #1 [41] is a well known regression problem, and we use a modified version that allows us to generate a variety of related concepts. Each instance x is a feature vector of length ten, with each component x_i drawn independently from the uniform distribution $[0, 1]$. The label for each instance is dependent on only the first five features:

$$\begin{aligned} y &= a_1 \cdot 10\sin(\pi(b_1x_1 + c_1) \cdot (b_2x_2 + c_2)) + \\ &\quad a_2 \cdot 20(b_3x_3 + c_3 - 0.5)^2 + a_3 \cdot 10(b_4x_4 + c_4) + \\ &\quad a_4 \cdot 5(b_5x_5 + c_5) + N(0, 1) \end{aligned}$$

where N is the normal distribution, and each a_i, b_i, and c_i is a fixed parameter. In the original Friedman #1 problem, each a_i and b_i is 1 while each c_i is 0, and we use these values when generating the target data set T_{target}. To generate each of the source data sets, we draw each a_i and b_i from $N(1, 0.1d)$ and each c_i from $N(0, 0.05d)$, where d is a parameter that controls how similar the source and target data sets are. Figure 8.1 illustrates the target and three random sources. Here y is plotted as a function of z by setting each $x_i = z$.

We performed experiments using several values d and values of 1 and 5 for B (the number of source data sets), expecting that transfer would be most effective for smaller values of d and the larger value of B. For each value of d, we randomly generated 100 of each of the following: i) target training data sets (of varying sizes), ii) target testing sets (of size 10,000), and iii) groups of 5 source data sets (each of size 1000). Neural networks were chosen as the best base learner.

For $d = 1$, Figure 8.2 shows the results for one source, and Figure 8.3 shows the results for five sources. Results for other values of d were qualitatively similar. As expected, using transfer increased accuracy the most for lower values of d and higher values of B.

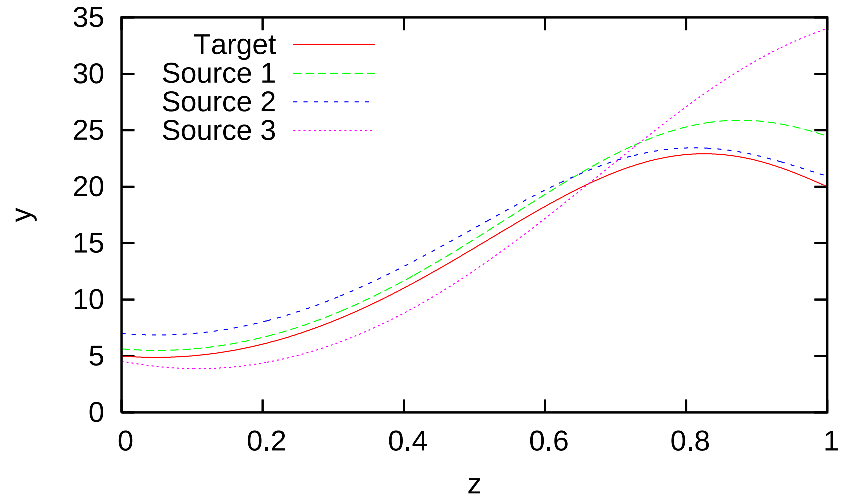

Figure 8.1: Illustration of Friedman #1

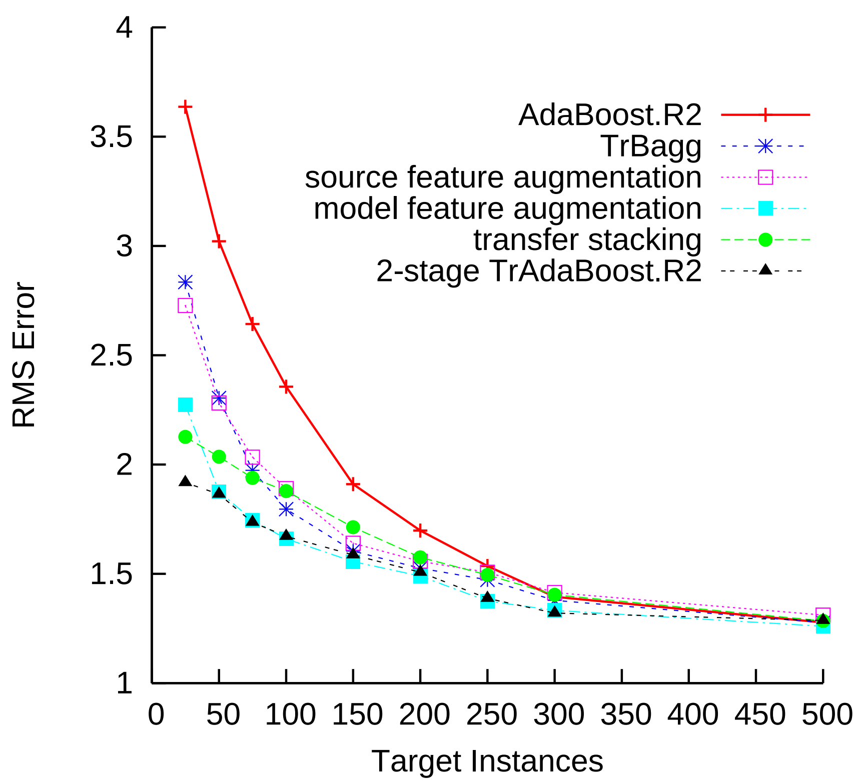

Figure 8.2: Friedman #1 with 1 source

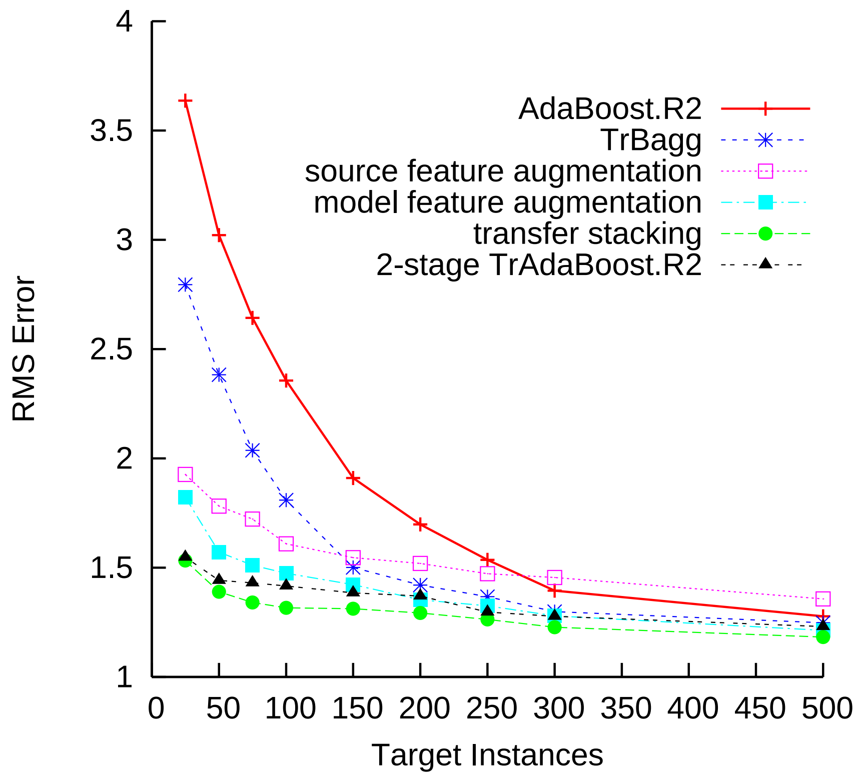

Figure 8.3: Friedman #1 with 5 sources

All transfer algorithms tested performed significantly better than AdaBoost.R2 for up to 200 target instances. Two-stage TrAdaBoost.R2, transfer stacking, and model feature augmentation were the most successful algorithms overall, and when five sources were available, all three algorithms were significantly better than AdaBoost.R2 for all points plotted. While TrBagg was also better than AdaBoost.R2 at most points, it did not appear to benefit as much as other algorithms from having five sources. TrBagg may fail to benefit when one source concept is especially similar to the target concept due to the low probability of the data set sampled at any iteration containing mostly instances from that source. Source feature augmentation actually falls significantly behind AdaBoost.R2 when there are sufficiently many target instances, especially with five sources, possibly due to the fact that source feature augmentation faces a more complicated learning problem

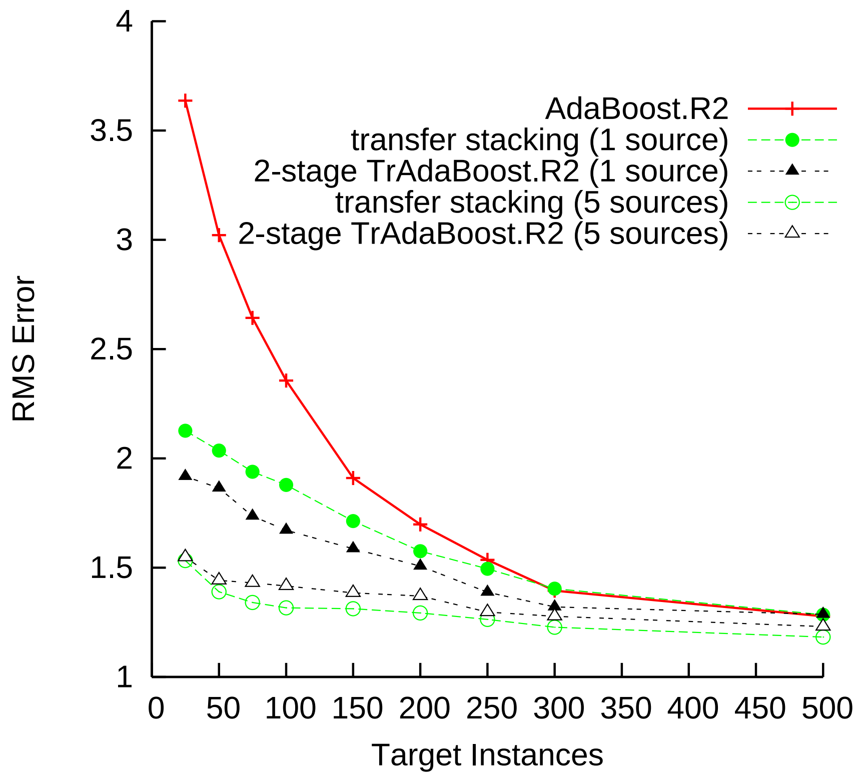

Figure 8.4: Friedman #1 with 1 and 5 sources

(with many more features).

To provide a clearer picture of how additional sources can improve transfer, Figure 8.4 shows the previous results for transfer stacking and two-stage TrAdaBoost.R2 with both one and five sources.

8.4.3 TAC Data Sets

The final four problems are prediction problems from the Trading Agent Competition, including three that have been discussed previously. There is one important difference between the way these problems were handled previously and the way they are handled here. In previous chapters concerning TAC, we considered learning from a certain number of completed TAC games and then testing the resulting models on a separate set of games (those that were presumably still in the future). Here, we approach learning using the standard machine learning setup used in this chapter: all instances (from all games) are combined into one data set, then learning curves are generated by randomly selecting subsets of these instances of different sizes. This process ensures that the training and testing sets are identically distributed. In the next chapter, I will discuss the difficulties presented by learning from complete games and show how the transfer algorithms can be modified to handle these difficulties.

The first problem comes from the TAC Travel scenario (not previously described). In TAC Travel, agents complete travel packages by bidding in simultaneous auctions for flights, hotels, and entertainment. We consider the problem of predicting the closing prices of hotel auctions given the current state of all auctions, represented by 51 features (as described in [88]). We use data from the 2006 competition final round as the target data, and data from the 2004 and 2005 final rounds as the source data sets. (In each year's final round, all games consisted of the same agents. Between years, however, the identities of some agents changed, while other agents improved significantly from year to year)

The next two problems are the computer and component price prediction problems from TAC SCM discussed in Chapter 5 (in Sections 5.1 and 5.2, respectively). For data, we use the data sets R1, R2, and R3 (generated using agent binaries as described in Section 5.4) as source sets, and the data from the 16 games of the final round of the 2006 competition as the target data set.

The final problem is the one discussed in Chapter 7, that of predicting the probability of a TAC AA advertiser placing a bid above a certain amount. Here we use data for epflagent as the target set and data for AstonTAC and Schlemazl1 as the source sets. This problem poses a challenge not seen in any other experiment: boosting is ineffective. In fact, when learning from target data alone, boosting

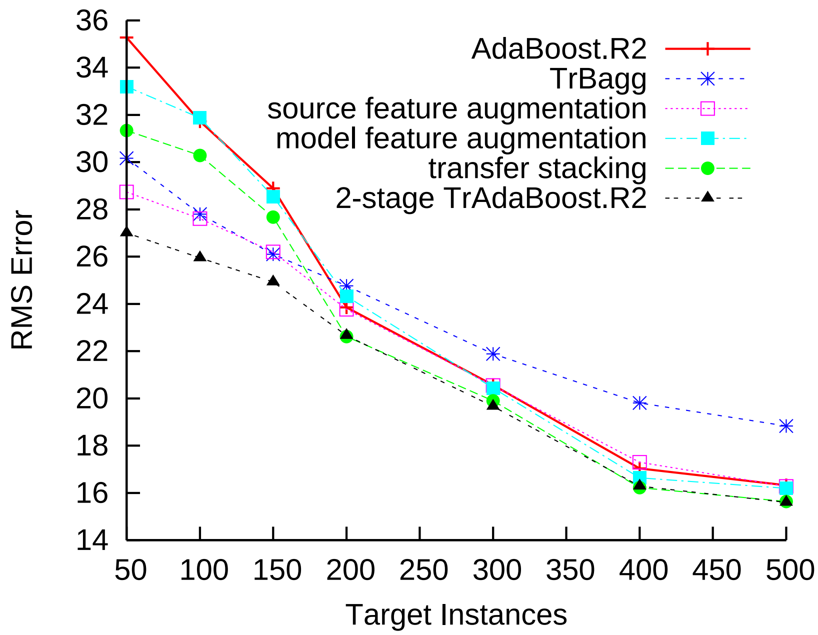

Figure 8.5: TAC Travel

actually results in higher errors. Recall that in this problem each instance is labeled with 1 or 0, and thus this could be seen as a binary classification problem in which we are interested in class membership probabilities. We treat this problem as a regression problem because that results in better accuracy on both predicted probabilities and the actual bid estimates we are ultimately interested in. However, applying AdaBoost.R2 in this case may have the effect of increasing the weights of instances representing low-probability events, which would lead to inaccurate probability predictions. In any case, for this problem, instead of boosting each algorithm, we perform bagging (again with 30 iterations), which is still effective. Two-stage TrAdaBoost.R2 is especially hurt by the failure of boosting, however, as it uses boosting in both stages. Although it is straightforward to replace boosting with bagging in the second stage, the first stage depends on boosting to select the most relevant source instances. Thus, for this problem, we replace two-stage TrAdaBoost.R2 with the best source weighting algorithm.

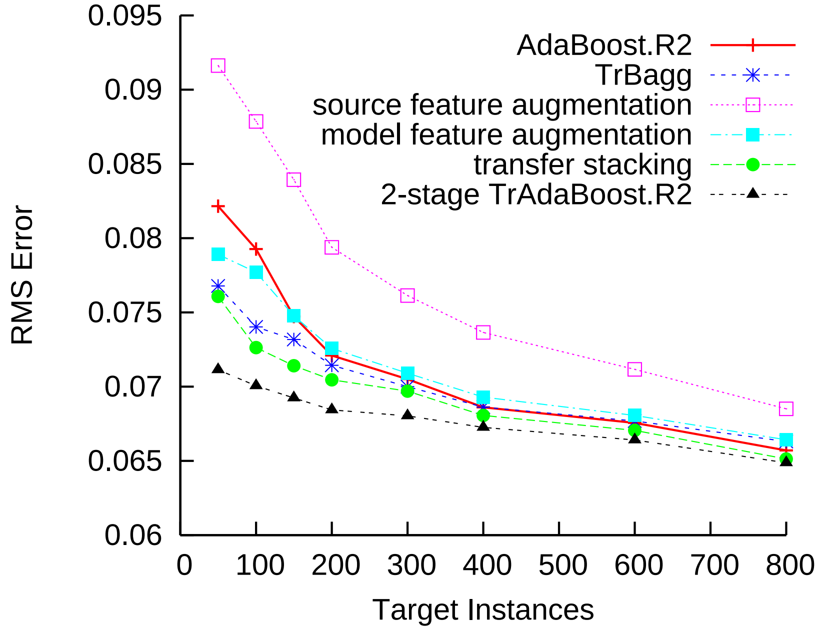

Figure 8.6: TAC SCM computers

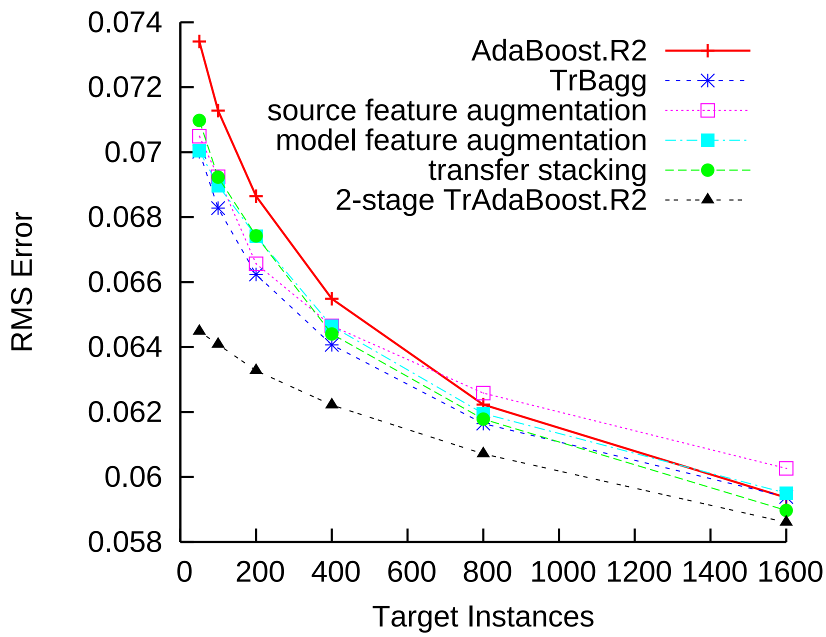

Figure 8.7: TAC SCM components

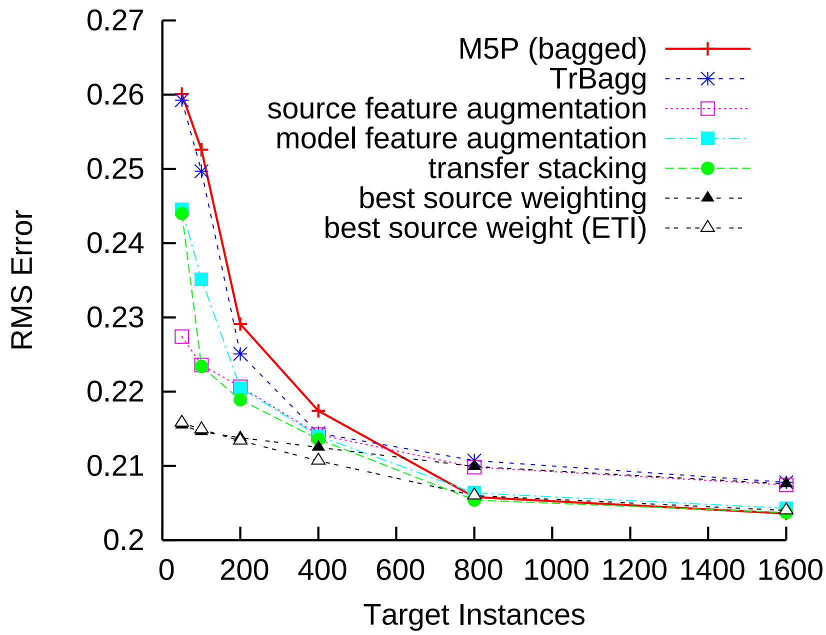

Figure 8.8: TAC AA bid probability

For all four problems, M5P model trees were chosen as the best base learner. Results are shown in Figures 8.5 through 8.8. For the three problems on which it is applicable, two-stage TrAdaBoost.R2 is the clear winner, as it performs particularly well when little target data is available, has significantly lower error than AdaBoost.R2 at all points, and never falls behind any other transfer algorithm. Performance of other algorithms is mixed, with only transfer stacking consistently performing as well as or better than AdaBoost.R2 on all four problems.

For the TAC AA problem, the best source weighting algorithm works surprisingly well when there is little target data available, but falls behind learning from target data alone when there is more target data. Upon closer inspection, in these cases the algorithm is correctly determining that little or no weight should be given to the source data, meaning that the results should be nearly identical to learning from target data alone. It turns out that the difference is in the size of the set of instances sampled at each bagging iteration. The size used is set to the total number of training instances, which is much larger when source data is mixed with target data. However, when the source instances receive little weight, most of the instances sampled will be target instances, each of which may be sampled several times. Using a larger bagging set thus reduces the expected difference between each set sampled, lessening the effect of bagging. Setting the size of the bagging set to be equal to the number of target instances causes the best source weighting algorithm to perform identically to learning from target data alone when many target instances are available, but harms its performance when few target instances are available. The solution is to set the bagging set size to be the number of *effective training instances*. Recall that in the tth step (out of s steps) of Algorithm 2, each target instance received a weight of 1 while each source instance received a weight of $1 - t/s$. The number of effective training instances at step t is thus $|T_{target}| + (1 - t/s)|T_{source}|$. Thus, as the weighting of source instances decreases, the size of the bagging set decreases. The results of using this modification are shown in Figure 8.8 (abbreviated as ETI).

8.4.4 Additional Experiments with Two-Stage TrAdaBoost.R2

In our final set of experiments, we took a closer look into the performance of two-stage TrAdaBoost.R2 due to the fact that this novel algorithm was consistently among the best. Two-stage TrAdaBoost.R2 differs from the best source weighting algorithm (when boosted) in two ways. In the first stage, two-stage TrAdaBoost.R2 reduces source weights through boosting (reducing weights the most on those instances where error is highest) rather than uniformly. Then in the second stage, it holds the source weights constant when building the final model through boosting, whereas the best source weighting algorithm treats all instance weights in the same way during boosting. To determine how much each difference contributes to the better performance of two-stage TrAdaBoost.R2, we tested four algorithms, one with each approach at each stage. Figures 8.9 through 8.11 show the results

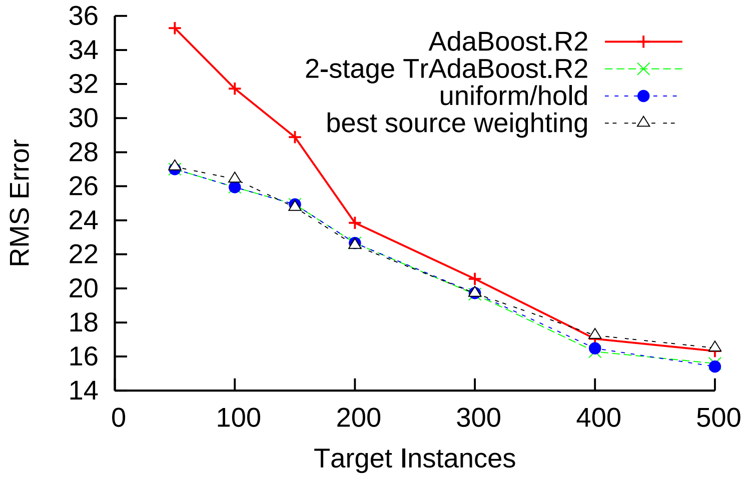

Figure 8.9: Two-stage alternatives, TAC Travel

on the three TAC problems for which boosting is effective. Here *uniform/hold* represents the algorithm that reduces source weights uniformly in the first stage but holds them constant in the second. The opposite algorithm that boosts source weights in the first stage but does not hold them in the second performs similarly to best source weighting and is not shown.

The surprising result of this experiment is that the method used to reduce source weights in the first stage has no significant effect on performance, at least for these three problems. For the two problems where best source weighting performs much worse than two-stage TrAdaBoost.R2, the difference is solely due to the change in the second stage.

To get a better idea of what is happening in the first stage, we plotted the weight given to source instances (as determined by cross validation in the first stage) as a function of the number of target instances available. Figures 8.12 through 8.14 show these weights for both the default two-stage TrAdaBoost.R2 and the uniform/hold version. Here, the source weight indicates how much weight the average source instance is given relative to each target instance–1 represents all instances being weighted equally, while 0 represents only target instances receiving

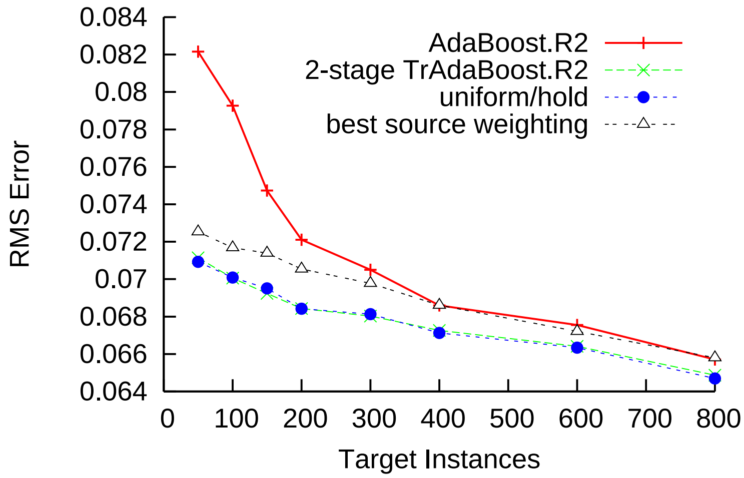

Figure 8.10: Two-stage alternatives, TAC SCM computers

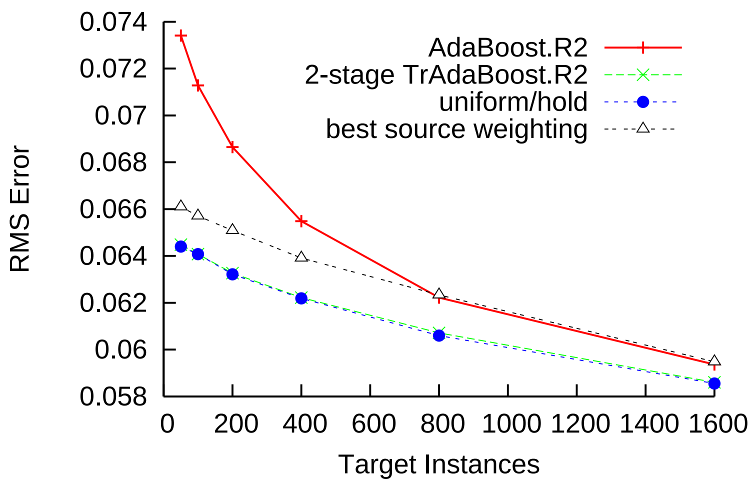

Figure 8.11: Two-stage alternatives, TAC SCM components

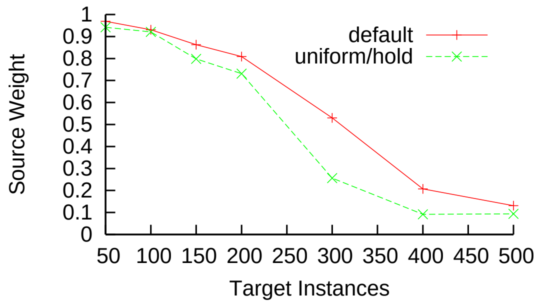

Figure 8.12: Source weight, TAC Travel

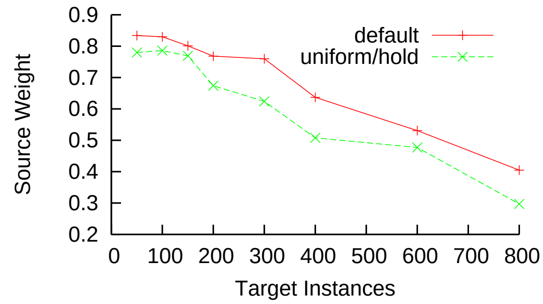

Figure 8.13: Source weight, TAC SCM computers

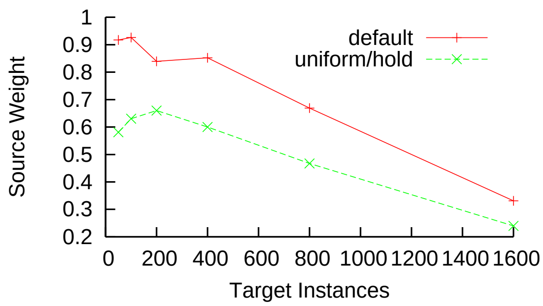

Figure 8.14: Source weight, TAC SCM components

weight. Interestingly, the default version of the algorithm always gives more weight to source instances than the uniform/hold version. In the default version, it is the least relevant source instances (in theory) that have their weights lowered the most, and so it may be optimal to use a higher average source weight overall to better benefit from those source instances that are relevant. This result suggests that using boosting in the first stage is in fact effective at identifying relevant source instances. For the three problems studied here, it appears that the uniform/hold version of the algorithm can compensate by using a lower overall source weighting, but that might not always be the case. In fact, in Section 9.4 of the next chapter, I present cases where the default version *does* outperform the uniform/hold version.

8.5 Summary

This chapter addressed the general problem of transfer learning for regression problems. A number of algorithms for regression transfer were presented, and experiments were performed in a wide variety of domains. Most of these algorithms can be broadly classified as either making use of models trained on source data or directly using the source data itself. Algorithms from the latter category appeared to do better overall. Two-stage TrAdaBoost.R2 was especially successful, as it typically dominated other algorithms on problems where boosting was effective, and is thus a significant contribution of this dissertation. Where boosting was not effective, simply finding the best source weighting appeared to be a reasonable substitute. Algorithms using source models also had some success, with transfer stacking appearing to be the most consistent of this group.

For the purposes of this dissertation, the most important result from this chapter is the fact that successful transfer is quite possible in a variety of regression problems, including several taken from TAC domains. In the next chapter, I will take a more in-depth look at applying the algorithms of this chapter to TAC.

Chapter 9

Using Past Game Experience in TAC

The previous chapter presented a number of algorithms for regression transfer. In this chapter, I take a closer look at the application of transfer learning within TAC and explore how the nature of the source and target data affects the performance of transfer algorithms. Throughout most of the chapter, results are presented in groups of three parallel experiments conducted on three data sets with each group testing data sets of a particular nature. The data sets used are taken from the TAC SCM computer price and component price problems of Chapter 5 (referred to as SCM-CPU and SCM-CPO) and the TAC AA bid probability data set of Chapter 7 (referred to as AA). For data sets from TAC SCM, the algorithms tested are AdaBoost.R2 using target data only (as a non-transfer baseline), 2-stage TrAdaBoost.R2 (the most successful algorithm from Chapter 8), and transfer stacking boosted with AdaBoost.R2 (as a representative of algorithms using source models). For the TAC AA data sets, the respective algorithms are bagging using target data, best source weight with bagging (with bag size set to the number of effective training instances), and bagged transfer stacking. For this chapter, algorithms will be abbreviated as `2ST` (for 2-Stage TrAdaBoost.R2), `TS` (for either bagged or boosted transfer stacking), and `BSW` (for best source weight with bagging). M5P model trees are used as the base learner in each case. For algorithms using source data, only 2000 source instances are used to speed learning; increasing this amount does not improve performance in most cases. The source models used in transfer stacking are learned from the full source data set using AdaBoost.R2. Results are averaged over 30 runs. In general, when one algorithm appears to clearly outperform another, the result is statistically significant (at least up to a certain target data set size), and so significance results are mentioned only where significance may be unclear. Results said to be significant are statistically significant ($p < .05$) according to paired t-tests.

9.1 Single Sources of Varying Similarity

The first aspect of source data to explore is the similarity of the (single) source to the target. For the next several sections, the `R1` data set will be used as the target for experiments on both TAC SCM problems, and Schlemazl2 will be used as the target for TAC AA experiments. (These were chosen as fairly typical data sets, and limited experimentation using other target data sets appeared to produce similar results. See Sections 5.4.1 and 7.2.1 for descriptions of all data sets.) The first group of experiments involves the least similar source data sets: `TT10` for SCM-CPU, `C06` for SCM-CPO, and MetroClick for AA. These will be referred to as *poor sources*. (See Tables 5.19, 5.20, and 7.4 for information on similarity between data sets.) Figures 9.1 through 9.3 show the results of all algorithms and of the source models. Even with such poor sources, `2ST` is highly effective on both SCM problems. This result is especially impressive for SCM-CPO, where a very accurate model can be learned with only 50 target instances even though `C06` is an extremely dissimilar data set. `BSW` for AA is not nearly as effective. `TS` is only helpful on SCM-CPU, likely due to the fact that `TT10` is the least dissimilar of the sources in this group of experiments.

The next group of experiments considers the opposite case where the source data is very similar to the target data. The source data sets used are `R2` for SCM-CPU, `R4` for SCM-CPO, and Schlemazl1 for AA. These will be referred to as *good sources*. Figures 9.4 through 9.6 show the results of all algorithms and of the source models. Not surprisingly, all transfer algorithms perform better than in the previous

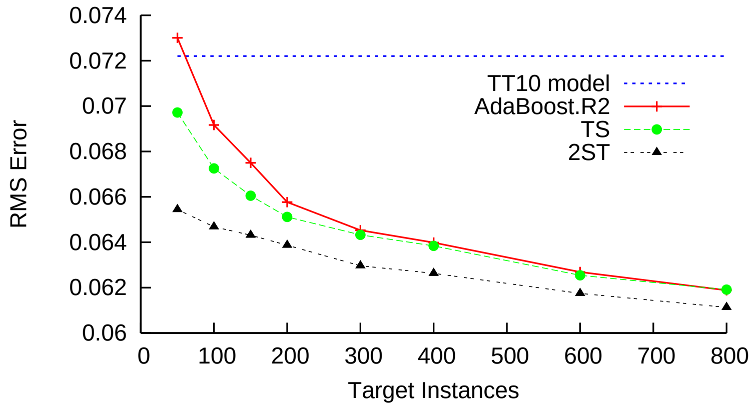

Figure 9.1: One poor source, SCM-CPU

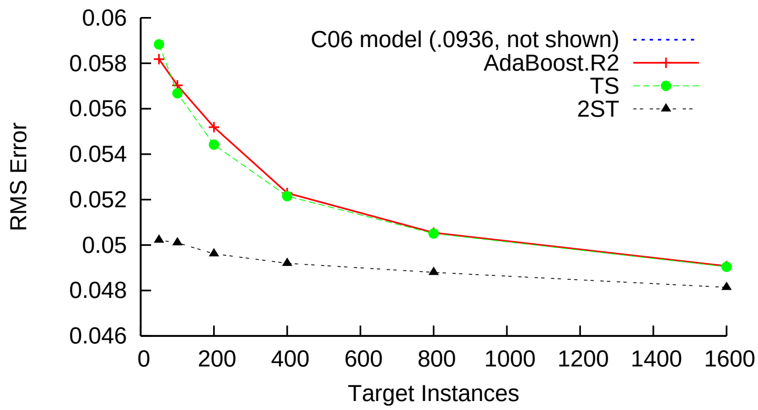

Figure 9.2: One poor source, SCM-CPO

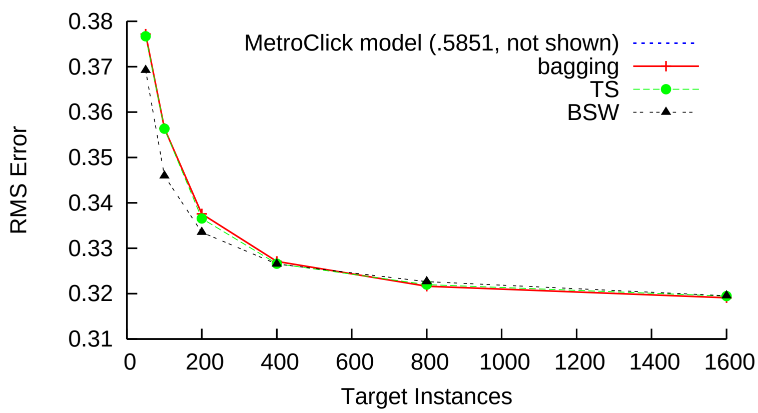

Figure 9.3: One poor source, AA

group of experiments. `2ST` remains better than `TS` overall on the SCM problems, although `TS` eventually catches up on SCM-CPO. `TS` also eventually surpasses `BSW` on AA. For all three problems, however, `TS` initially results in a higher error than simply using the source model, meaning that it is unable to correctly identify the best combination of models (i.e., giving all weight to the source model) when target data is limited.

The final group of experiments in this section considers the case where the source data is only moderately similar to the target data. The source data sets used are `F10` for SCM-CPU, `TT10` for SCM-CPO, and epfl for AA. These will be referred to as *medium sources.* Figures 9.7 through 9.9 show the results of all algorithms and of the source models. For `TS`, performance lies somewhere between the performance when good and poor sources are used, as expected. For `2ST` and `BSW`, however, the results are not so straightforward, as seen in Figures 9.10 through 9.12. In fact, on both SCM problems, using the poor source significantly outperforms using the medium source for certain amounts of target data. One possible explanation for this result is that while the poor source contains instances that are less relevant overall to the target, it may contain some instances that are highly relevant, and `2ST` is capable of identifying and using these instances.

9.2 Source and Target are Identical

Whereas the previous section considered single sources that were similar to the target, this section explores the possibility that the source and target distributions are identical. Of course, if the source and target are known to be identical, then the obvious solution is to combine the source and target data, or simply to use an existing model that has been trained on the source if the source data is sufficiently plentiful. If it is even suspected that the source and target might be identical, then it is likely worthwhile to check if this is the case using an appropriate

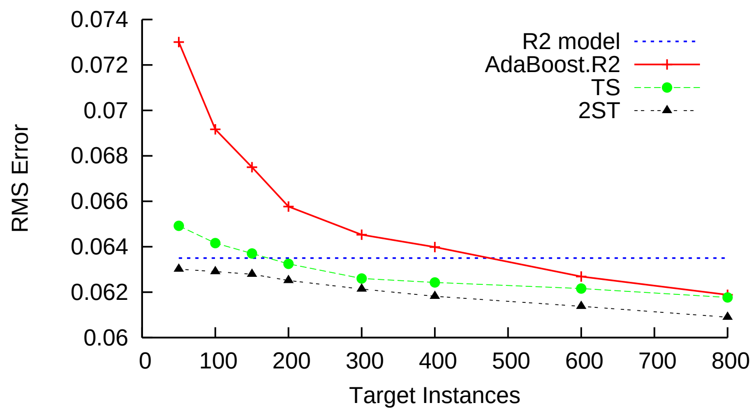

Figure 9.4: One good source, SCM-CPU

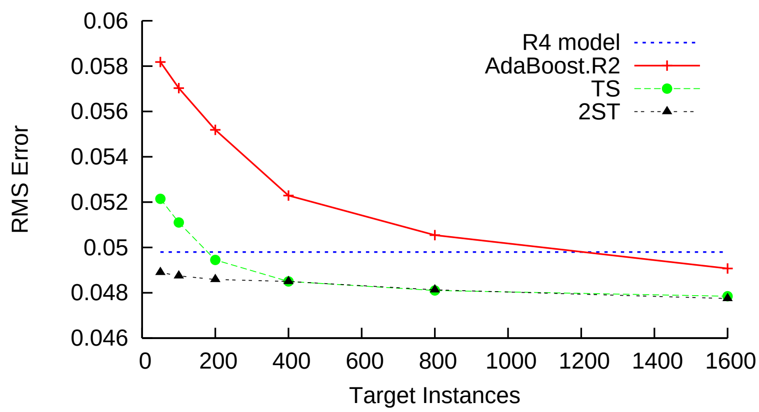

Figure 9.5: One good source, SCM-CPO

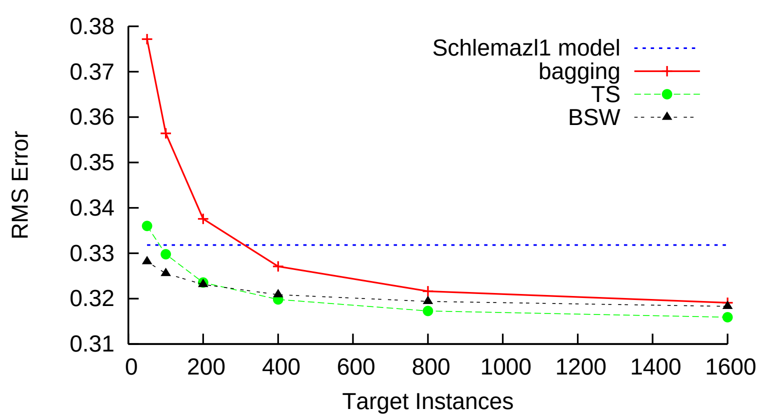

Figure 9.6: One good source, AA

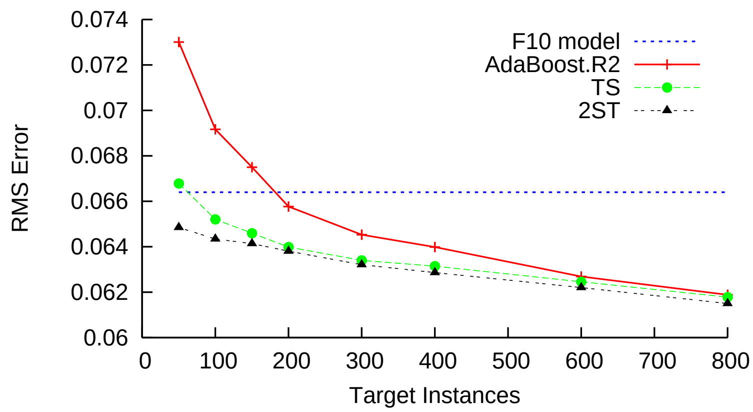

Figure 9.7: One medium source, SCM-CPU

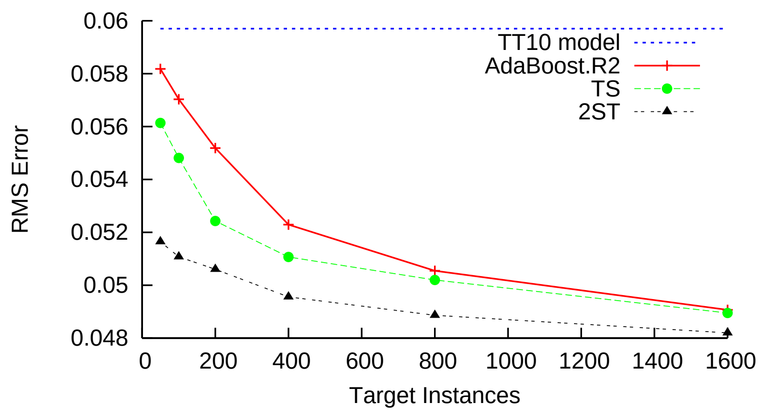

Figure 9.8: One medium source, SCM-CPO

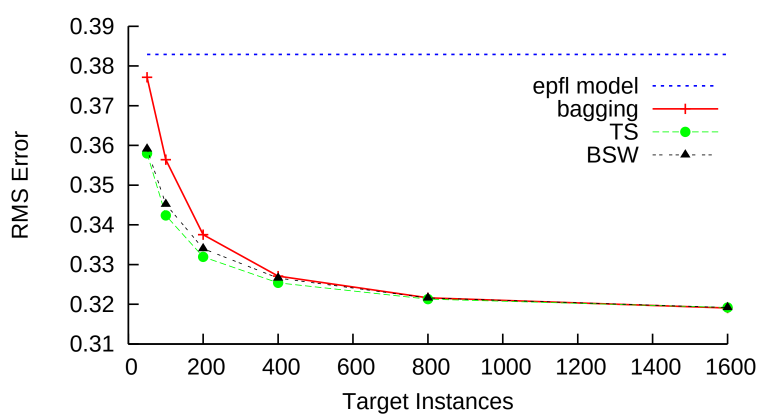

Figure 9.9: One medium source, AA

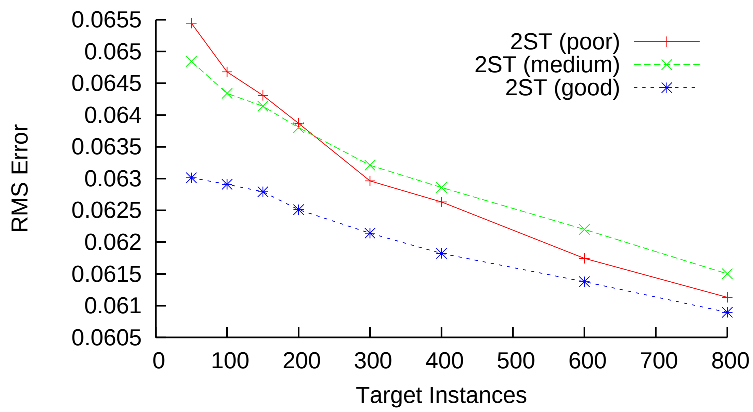

Figure 9.10: `2ST` with each source, SCM-CPU

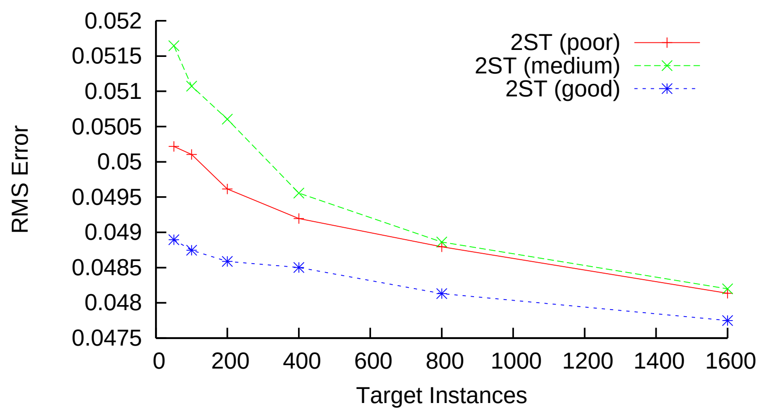

Figure 9.11: `2ST` with each source, SCM-CPO

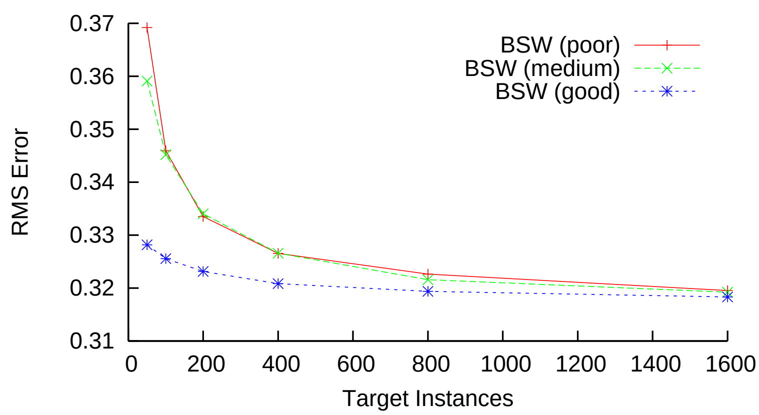

Figure 9.12: `BSW` with each source, AA

statistical test. Similarly, if there are multiple sources and it is believed that one of the sources is identical to the target, then the simple method of finding the best source model (described in Section 8.2.2) is probably the right approach. Nevertheless, it is worth exploring the performance of the algorithms under consideration in this chapter in case this situation arises unexpectedly. In addition, the results of this section are probably similar to the results when the source and target are not identical, but extremely similar.

The target data sets used for this group of experiments are the same as the targets of the previous section. Figures 9.13 through 9.15 show the results when the source data comes from the same data set (with no overlap of instances), including the results for the model trained only on the source data. Not surprisingly, the accuracy of the transfer algorithms is initially significantly worse than the source model. `TS` (which produces a linear combination of the source model and a model trained on the target data) quickly learns to put most weight on the source model, and thus catches up to the source model's performance. `2ST` and `BSW` remain significantly worse than the source model, however, despite the fact that they both correctly learn to give source instances about as much weight as target instances. There are two reasons for this result. First, as mentioned previously, both of these algorithms train using only 2000 source instances to reduce training time. This number is sufficient when the source differs from the target, but additional instances are helpful when the source and target are the same. Training with all source instances results in `BSW` performing just as well as the source model on AA, which should be expected since `BSW` treats target instances and source instances (once weighted) the same. On the other hand, `2ST` remains much less accurate than the source model no matter how much source data is used for training due to the fact that in its second stage, `2ST` fixes the weights of source instances during boosting. If boosting is helpful, as is the case for both SCM problems, then `2ST` is clearly at a disadvantage despite having access to the same training data. (Recall that

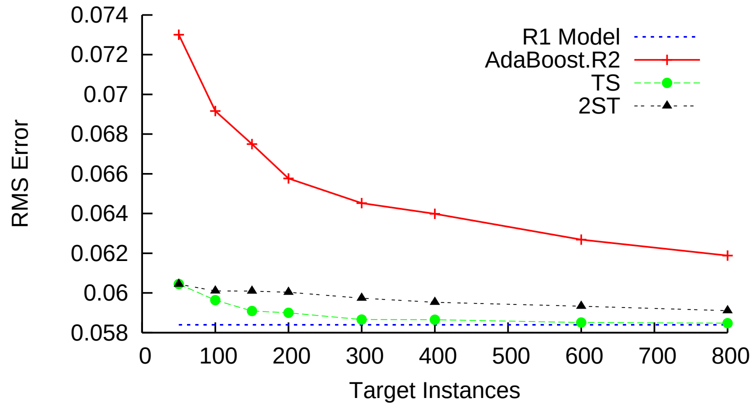

Figure 9.13: Source equals target, SCM-CPU

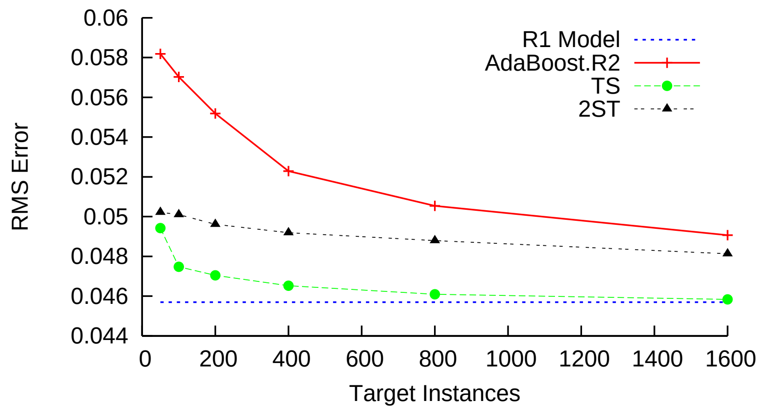

Figure 9.14: Source equals target, SCM-CPO

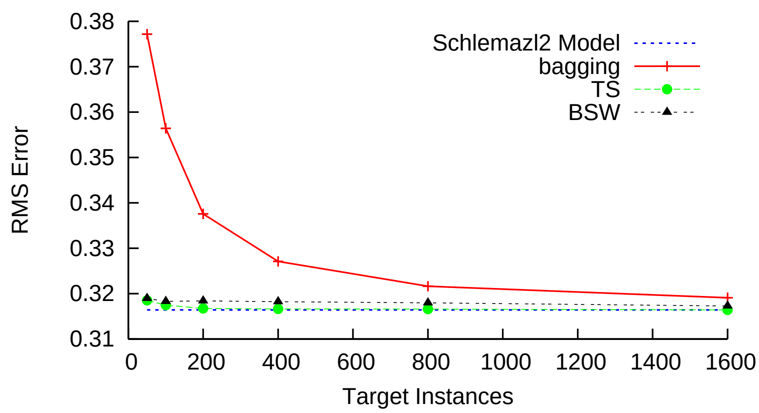

Figure 9.15: Source equals target, AA

the source model is trained using AdaBoost.R2.) For this reason, it is especially important to check whether the source and target distributions are identical before using `2ST`.

9.3 Multiple Sources

The next aspect of source data to consider is the number of sources. If multiple sources exist, is using more sources always better? This section will explore this question using the same poor, medium, and good sources used for each problem in the previous section.

The first group of experiments considers using either one good source or two sources: one good and one poor. Figures 9.16 through 9.18 show the results of the transfer algorithms on each problem. Similarly, the second group of experiments considers adding a medium source to a good source. Figures 9.19 through 9.21 show the results. Results in both cases are similar.

For `TS`, performance when using two sources is significantly worse on most problems when few target instances are available. This result is not surprising, as adding more source models increases the number of parameters to tune, thus increasing the risk of overfitting. Performance from using two sources does catch up quickly as more target instances become available. In fact, when a few hundred target instances are available for SCM-CPU, performance is better when using two sources (but not quite significantly better, in most cases).

For `2ST`, adding a poor source to a good source appears to result in a slight improvement in accuracy, while adding a medium source does not. This result is consistent with Figures 9.10 and 9.11, where a single poor source was sometimes better than a single medium source. Note that none of the observed differences are significant, however. For `BSW` on AA, adding a second source results in a significant improvement in accuracy when the number of target instances is small, but the second source actually decreases accuracy (though not significantly) when there are many target instances.

The second group of experiments considers using all three sources. Figures 9.22 through 9.24 show the results of the transfer algorithms on each problem. For `TS`, the results are similar to the results of the experiments involving two sources, only more pronounced. With only 50 target instances, the accuracy when using all sources falls even further behind the accuracy from using just the good source or using two sources. Again, the increase in tunable parameters appears to lead to overfitting. When a few hundred target instances are available for SCM-CPU, however, performance is significantly better when using all sources than when using one or two sources.

For `2ST` on both SCM problems, when up to several hundred target instances are available, using all three sources results in a significant improvement in performance over using one or two sources. For `BSW`, using all three sources results in essentially the same performance as using only the good source.

Overall, deciding how to deal with multiple sources is simplest with `2ST`. It appears that adding more sources will not notably harm performance and may significantly improve it, so the best approach is probably to just use all available sources (although this observation may not hold if the number of sources is much greater than the two or three considered here). When there are sufficiently many target instances, `TS` also appears able to properly handle multiple source models, but care should be taken when the number of target instances is small. In such cases, the best approach might be to use only the model from the source that is most similar to the target. However, it should be noted that in a real application, we might not know which source this is. Unless some prior knowledge helps us to identify which source is most appropriate, we could choose the source model that has the lowest error on the target instances, but given the small number of target

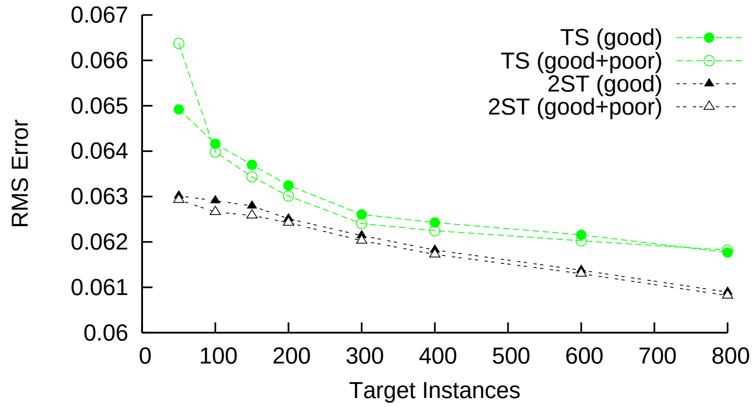

Figure 9.16: Adding a poor source to a good source, SCM-CPO

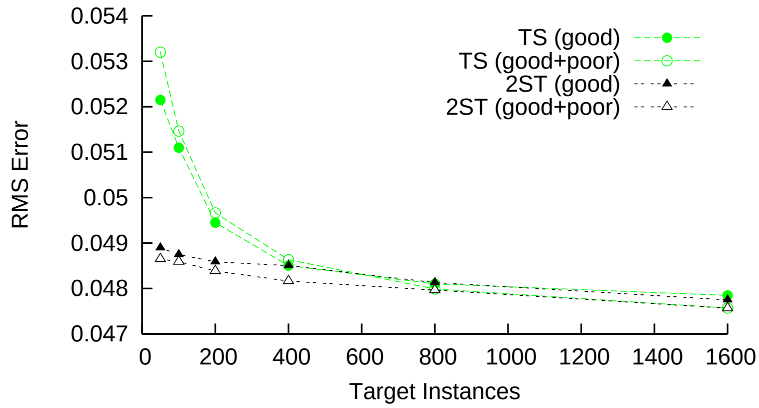

Figure 9.17: Adding a poor source to a good source, SCM-CPO

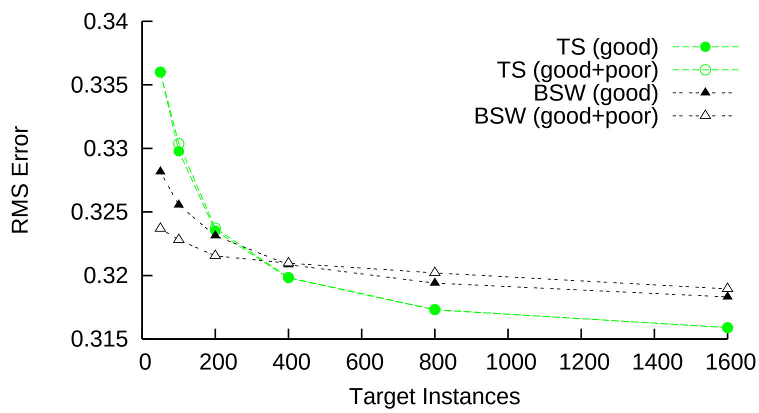

Figure 9.18: Adding a poor source to a good source, AA

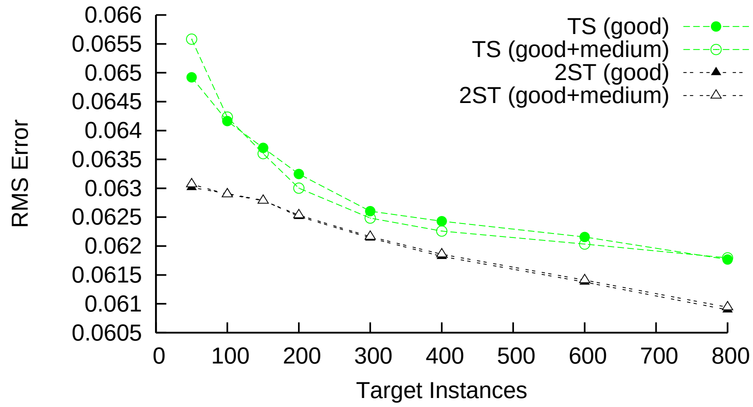

Figure 9.19: Adding a medium source to a good source, SCM-CPU

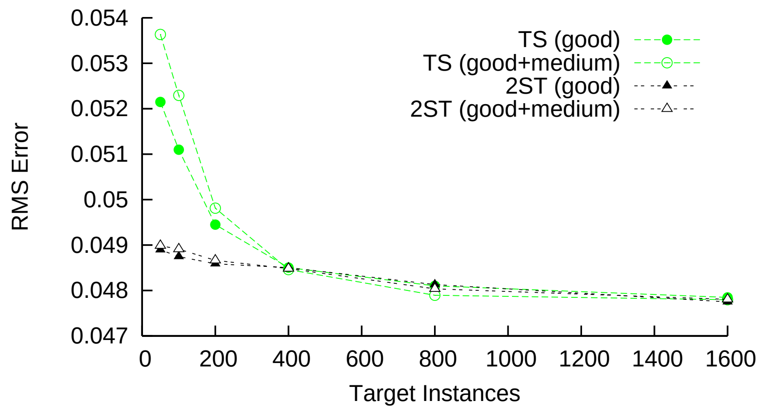

Figure 9.20: Adding a medium source to a good source, SCM-CPO

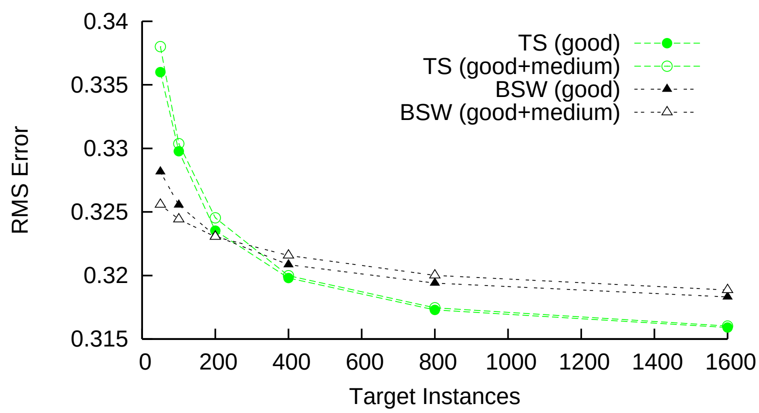

Figure 9.21: Adding a medium source to a good source, AA

instances we might not have much confidence in this choice. A better approach might be to use some type of regularization to encourage the use of fewer source models in the combination of models output by `TS`. Results for `BSW` are mixed, which is not surprising as `BSW` has no means of differentiating between different sources. Again, the best approach might be to use only the source believed to be most similar.

9.4 Source Weighting in `2ST`

I now briefly return to an issue discussed in the previous chapter. Recall that the experiments of Section 8.4.4 suggested that the method used to reduce the weights of source instances in the first stage of `2ST` might not matter. Specifically, for the problems considered, the accuracy of the model learned was the same whether source instances had their weights individually reduced through boosting (as is normal in `2ST`) or uniformly reduced across all instances. It turns out that in the experiments of the previous section, in which multiple sources were used, reducing weights though boosting is preferable. Figure 9.25 shows the results when both weighting methods are used on SCM-CPO with one good source and one poor source. Using boosting in the first stage was significantly better than using uniform weighting across all numbers of target instances tried. Results for other source combinations on both SCM problems were similar–using boosting usually performed better, and never performed worse, than using uniform weighting. This observation makes sense, since weighting source instances individually allows instances from a good source to generally receive more weight than instances from other sources.

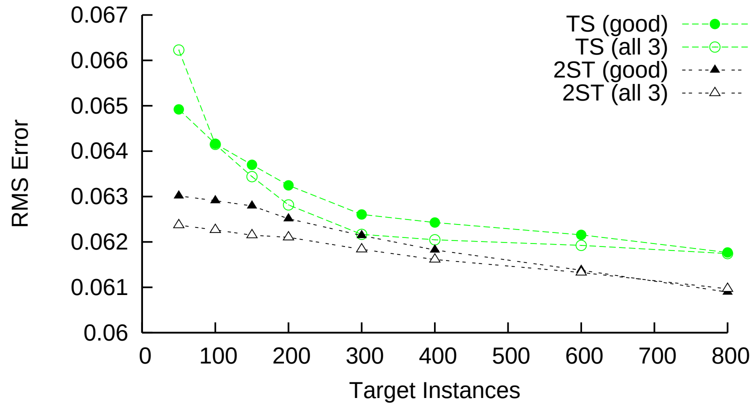

Figure 9.22: All 3 sources, SCM-CPU

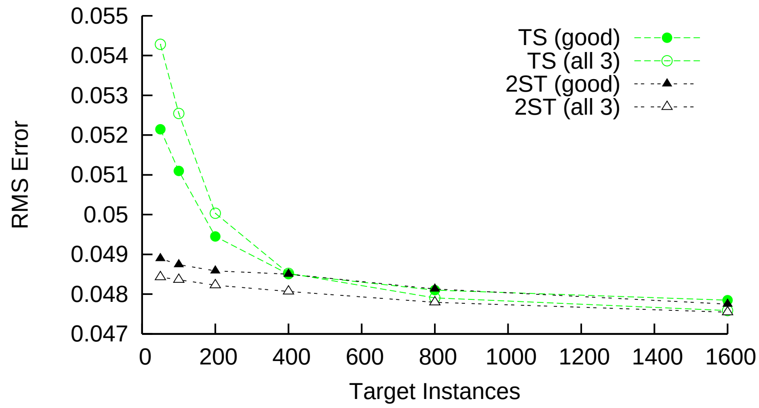

Figure 9.23: All 3 sources, SCM-CPO

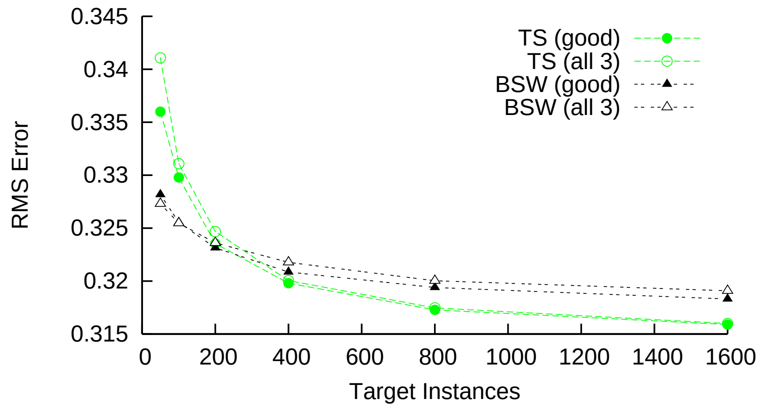

Figure 9.24: All 3 sources, AA

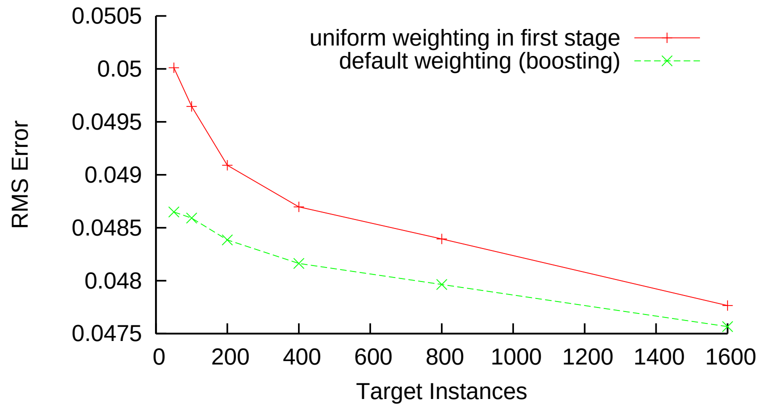

Figure 9.25: Uniform source weights in the first stage of `2ST`, SCM-CPO. One poor source and one good source used.

9.5 Limited Source Data

The experiments so far have assumed that we have ample source data (several thousand instances). This section looks at the success of transfer when source data is limited.

The first group of experiments considers the case where we have one good source (using the same data sets from the previous sections), but only 200 source instances. Figures 9.26 through 9.28 show the results of the transfer algorithms on each problem using both this limited source data and the full source data used previously. As expected, limiting the amount of source data hurts the accuracy of the models learned by all transfer algorithms. `TS` performs especially poorly, with no significant improvement over the non-transfer method at most points. This result is not surprising, as a model learned from a small amount of source data may not even fit the source concept well, let alone the target concept. The source models are thus given little weighting in the final combination of models output by `TS`, limiting the potential for transfer. `2ST` and `BSW` fare better, showing significant transfer at most points. The fact that these algorithms directly combine source and target instances appears to allow them to make effective use of even small amounts of source data. In any case, the performance of all three algorithms is never worse than the performance of the non-transfer algorithm, so when one source is available, it is likely worthwhile to attempt transfer no matter how few source instances there are.

The next group of experiments considers the case of two sources when only one source is limited to 200 instances. 200 instances from the good source are used again, and the full number of instances from the medium source are added. With `TS`, little weight is given to the relatively poor model learned on the good source, and so results are not significantly different from the results when only the medium source is used. This outcome suggests that the quantity of source data is not an important consideration when using `TS` with multiple sources, as `TS` can learn to ignore a source for which too few instances are available.

For `2ST` and `BSW`, recall that 2000 source instances are being used for training when the full amount of source data is available, and that by default all source and target instances receive the same initial weight. This means that the medium source receives ten times as much weight overall as the good source (with only 200 instances). One possible modification to these algorithms would be to give each source an equal overall weight (in this case, by multiplying the initial weight of each instance from the good source by ten). Figures 9.29 through 9.31 show the results for `2ST` and `BSW` when using the good source, the medium source, both sources with equal instance weights (the default), and both sources with ten times the weight on each good source instance. The most striking observation from these figures is that adding weight to the good source instances produces results that are usually significantly worse than using the default weighting, and often worse than using only the medium source. This observation tends to remain true even when the weights of good source instances are multiplied by a factor less than ten. Thus, in the case of multiple sources, it is likely best to simply weight each source instance equally, regardless of the relative sizes of the source data sets. For `2ST`, combining

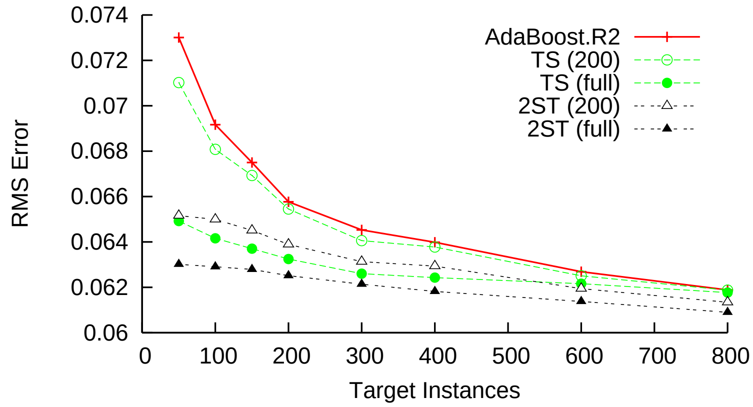

Figure 9.26: Limited good source, SCM-CPU

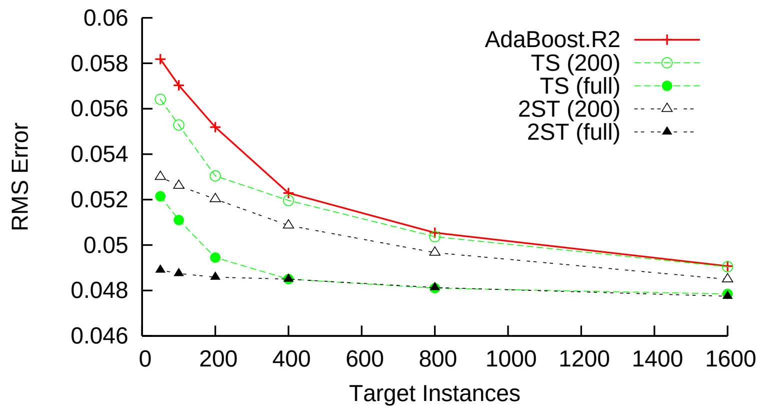

Figure 9.27: Limited good source, SCM-CPO

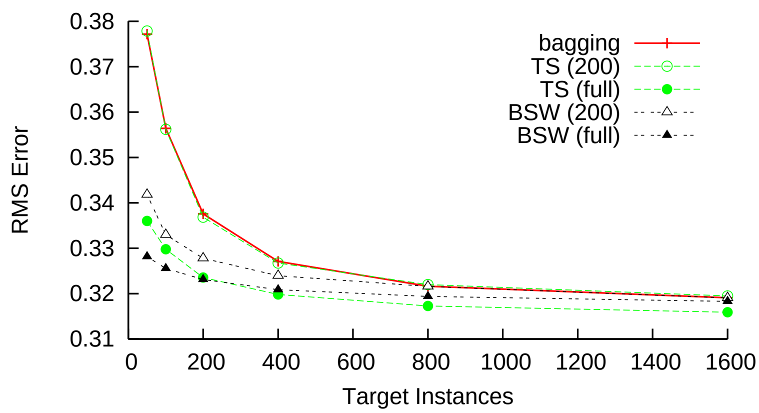

Figure 9.28: Limited good source, AA

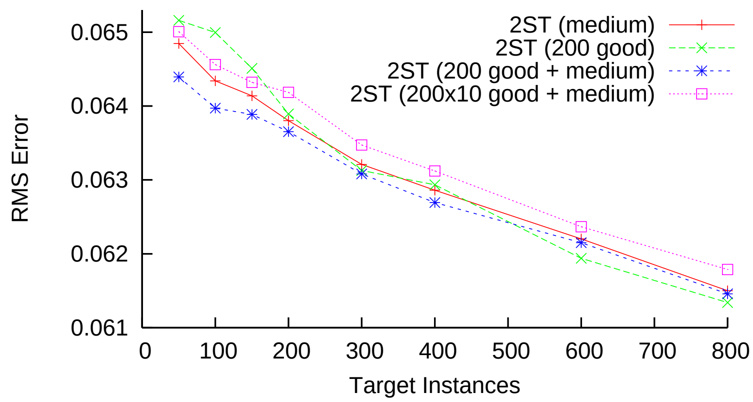

Figure 9.29: Limited good and full medium source, SCM-CPU

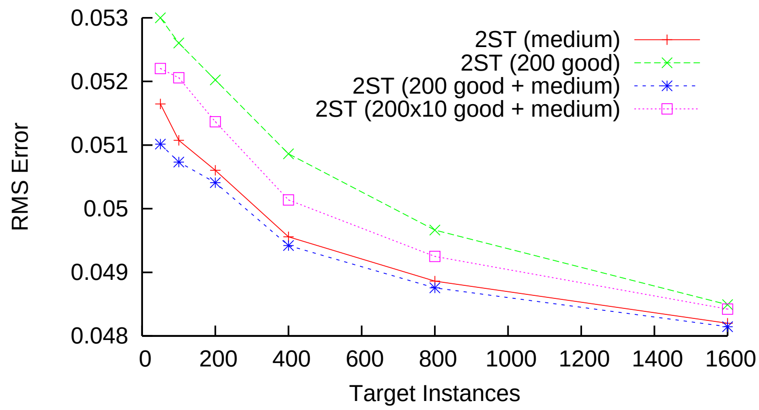

Figure 9.30: Limited good and full medium source, SCM-CPO

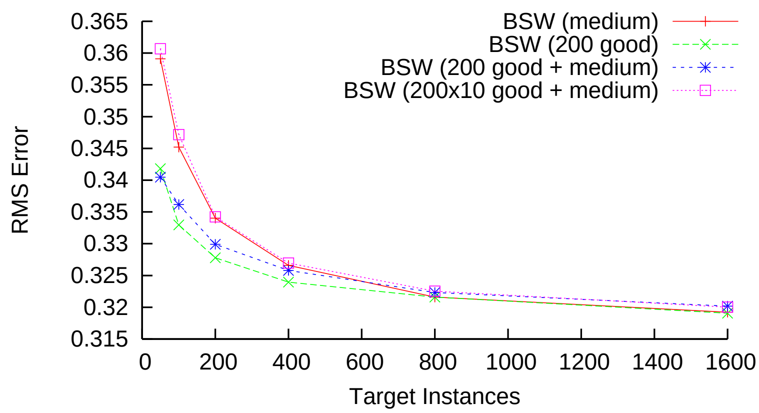

Figure 9.31: Limited good and full medium source, AA

both sources usually produces better results than using either source alone, and is never significantly worse. As was the case in the experiments of Section 9.3, it appears that the best strategy for `2ST` is to simply use all available sources. Once again, `BSW` appears to suffer from the inability to weight source instances separately, as the results from using both sources are slightly (but not significantly) worse than the results of using only the good source.

9.6 Learning from TAC Games

To this point, we have worked within a standard supervised learning setting where training data is independent and identically distributed (IID). Results have taken the form of learning curves over different numbers of target instances where these instances are individually drawn from a large data set. However, if we apply transfer learning in TAC, then the target training data comes from a number of completed games. Instances from the same game are not independent (e.g., if the price of one computer type rises on a given day, then the prices of other computer types are likely to rise as well), and instances from different games are not identically distributed (e.g., one SCM game may have higher customer demand levels than another). Similar difficulties arise in using data from real world markets. It is therefore important to test our transfer algorithms under TAC-like settings and consider what adjustments need to be made.

The main concern with non-IID data is that all three of our transfer algorithms rely on N-fold cross validation of the target training data for weighting either source models or source data. If the data in each fold is not independent, then the estimated error of a model trained on target data may be artificially low. As a result, `TS` might give too much weight to the model trained on target data, and `2ST` and `BSW` might give too little weight to source instances. One way to address this problem in the TAC setting is to treat data from each available game as a separate fold, since games are independent. Figures 9.32 through 9.34 show the results of using both the default cross validation and one-fold-per-game cross validation. Here one good source was used. In addition, 500 target instances per target game were used, as increasing this number did not provide any benefit. Note that at least two games are necessary for one-fold-per-game cross validation to be possible.

Using one-fold-per-game cross validation provides a clear benefit over the default cross validation in SCM-CPU, but there is usually no significant benefit on the other problems. One possible explanation for this result is that the prices of different components in TAC SCM and the bids on different queries in TAC AA are fairly uncorrelated, while the prices of different computers in TAC SCM tend to move together. Assuming that target instances from a single game are independent of each other may therefore be more unrealistic in SCM-CPU. Another interesting observation is that `TS` performs nearly as well as `2ST` overall (although `2ST` does remain significantly better on SCM-CPU). In the experiments of previous sections, it often took `TS` several hundred target instances to catch up to `2ST`, but in this setting, even a single game represents hundreds of target instances, so `2ST` may have less of an advantage in this situation. In any case, all three transfer algorithms with one-fold-per-game cross validation perform significantly better than the non-transfer algorithm, showing that successful transfer is possible in an actual TAC setting.

9.7 SCM Agent Performance

The next logical step is to test whether using transfer learning can actually improve agent performance. We now test this in a setting previously considered in Section 5.4.1. (We restrict our attention to TAC SCM, as this is the TAC scenario for which we have observed the clearest link between model accuracy and agent

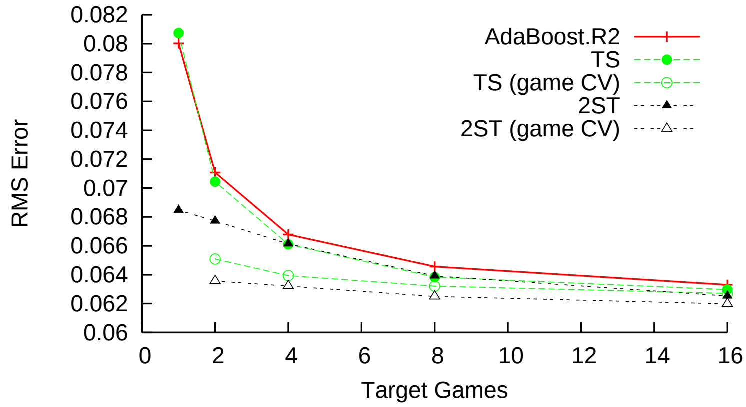

Figure 9.32: Learning from TAC games, SCM-CPU

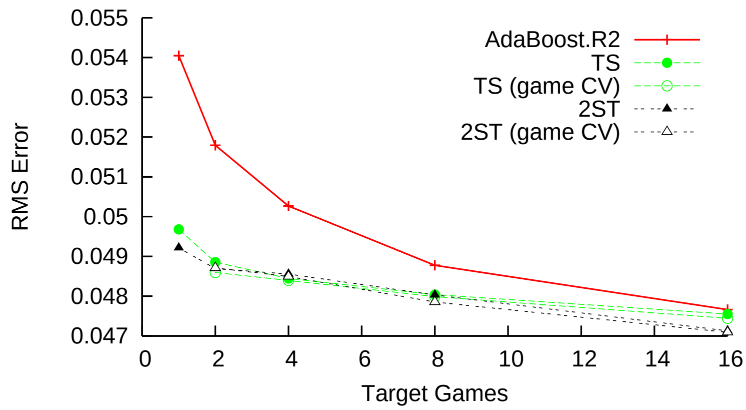

Figure 9.33: Learning from TAC games, SCM-CPO

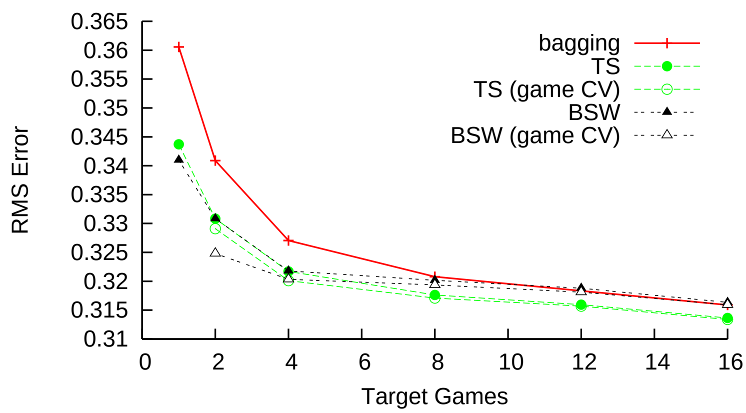

Figure 9.34: Learning from TAC games, AA

Target Games	*Change in score vs default model*	
	AdaBoost.R2	2ST *(game CV)*
4	-0.29	$+0.23$
8	$+0.14$	$+0.36$

Table 9.1: Results of using transfer in an SCM agent to learn computer and component models. Differences in scores (in millions) between an agent using default models and an agent using models learned online.

performance.) The experiments of that section showed that in TAC SCM, when one TacTex10 agent played against five other TacTex10 agents (the TT10 group), learning computer and component models that were specific to these agents could significantly improve the agent's score over an agent that used the default models (trained on the groups R1,R2,R3, and R4). When 30 games were available for training, the agent using both computer and component models trained on TT10 outperformed the agent using the default models by 1.12 million per game.

We now test the use of transfer learning in this setting. Table 9.1 shows the difference in score between an agent using models trained on 4 or 8 target games (TT10) and an agent using the default models. Both AdaBoost.R2 (no transfer) and 2ST (using the one-fold-per-game cross validation of the previous section) are used as learning algorithms. For source data, 2ST uses the same data used to train the default models. Results are averaged over 30 games. For both 4 and 8 games, 2ST results in a higher score than either using the default models or using AdaBoost.R2, showing that transfer learning would have been beneficial in this situation. It should be noted that these differences in scores are not statistically significant, however. Establishing significance (if possible) would likely require repeating these experiments over a large number of training sets.

9.8 Transfer in a Non-stationary Setting

The experiments so far have assumed that we are in a stationary setting, where competing agents never change their behavior. This section briefly considers an application of transfer learning to a non-stationary setting. One straightforward approach to dealing with a non-stationary setting is to use only the most recent data for training. The amount of data that should be used depends on the rate of change of the system and the amount of data needed for effective learning. Identifying the optimal cutoff point is beyond the scope of this chapter, but whatever this point is, any data before this point is effectively wasted. One possible approach to making use of this extra data is to treat it as a separate source and apply a transfer learning algorithm (possibly in conjunction with other sources). `2ST` is especially suited to this approach because it weights source instances individually, and can therefore learn to place less weight on more outdated instances.

To generate non-stationary behavior for training, we ran 30 SCM games with six TacTex10 agents. Over the 30 games, three tunable agent parameters gradually changed. First, we introduced a multiplier on predicted computer price changes that increased from 0 to 3, making the agent increasingly likely to either hoard or sell off computers. Second, we introduced an adjustment to predicted future component prices that increased from -0.5% to 0.8% per day before the requested due date, making the agent increasingly likely to place short term orders. Third, the size of certain long-term orders placed at the start of the game decreased from the full amount to one third that amount, further emphasizing short-term procurement. At the end of the 30 games, the final parameters were fixed, and 30 more games were run to generate testing data representing the final concept. We considered using the last N games from the training set, where N ranged from 4 to 28. Figures 9.35 and 9.36 show the results for AdaBoost.R2 and `2ST` (treating the remaining data as a source, and using one-fold-per-game cross validation). Again, 500 instances per game were used, both as source and target. For any choice of

Figure 9.35: Non-stationary setting, SCM-CPU

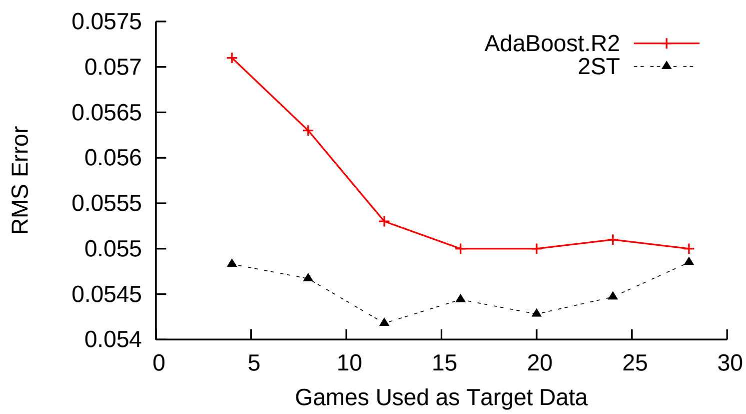

Figure 9.36: Non-stationary setting, SCM-CPO

N, using the additional games as source resulted in a clear reduction in error. In addition, choosing N correctly appears to be less critical when using the additional games, as the learning curves for `2ST` are somewhat shallower. These results suggest that transfer learning can be an effective method of dealing with non-stationary data. Note that testing for statistical significance is not possible as these results involve a single run of 30 games.

9.9 Summary

This chapter explored the use of transfer algorithms in different TAC settings. The experiments presented were designed to show how the performance of the different transfer algorithms is affected by the nature of the source and target data. As was the case in the previous chapter, when boosting is effective (in SCM-CPU and SCM-CPO), `2ST` generally performed best. In addition, the results suggest that a straightforward application of `2ST` is very robust to different situations. `2ST` produced effective transfer even when the source concept was dissimilar to the target concept or when only a small amount of source data was available. When multiple sources were available, simply combining all source data produced results that were as good as, and sometimes better than, using only the best source. With one-fold-per-game cross validation, `2ST` proved successful at learning models online from completed games in a realistic TAC setting, and the results of using the resulting models in an agent were promising. `2ST` can even be used as a means of learning from non-stationary data. The only caveat is that `2ST` may not be the most effective algorithm when the source concept is extremely similar to (or the same as) the target concept. Results for `TS` and `BSW` were more mixed, and a number of potential problems with these algorithms were identified. Nevertheless, both were very effective in certain situations. In particular, `TS` performed very well when learning from full games in the realistic TAC setting, and its advantage over `2ST` in running time would be particularly useful in this situation.

Chapter 10

Related Work

There is a considerable amount of existing work that is related to either the market domains, agent designs, or learning approaches described in this dissertation. The most relevant references have been discussed where appropriate in the preceding chapters, but to improve the flow of the text, the remainder has been left until now. This chapter places the contributions of this dissertation in the context of this existing work and includes comparisons with alternative approaches to similar problems. The organization of this chapter is parallel to the organization of the dissertation. Work related to adaptive auction mechanisms (Chapters 2 and 3) is discussed first, then work on the two TAC domains (supply chain management in Chapters 4 and 5, and ad auctions in Chapters 6 and 7), and finally work on transfer learning (Chapters 8 and 9).

10.1 Adaptive Auction Mechanisms

Auction theory (see e.g. [61] and [59] for an overview) has a long history, beginning with the work of Vickrey [106] and continuing through recent decades (e.g., [70, 73, 82]). The growth of electronic commerce and Internet auctions in recent years has led to a renewed interest in auction theory by providing new scenarios (such as the generalized second price auctions used in keyword auctions [38]) and behaviors (such as sniping, the act of waiting until the last second to place a bid [81]) to analyze. While much of this work has traditionally been game-theoretic, the work most relevant to this dissertation is that which is empirical in nature, ranging from experiments with new auction mechanisms in simulation to efforts to model (potentially irrational) bidder behavior from the large quantity of real online bidding data that has become available in recent years.

When extensive historical data on past auctions of identical items is available (as is the case with eBay), and when past patterns are likely to persist, it may be possible for an auctioneer to estimate the optimal auction parameters by analyzing this data. A number of tools exist for this purpose. For example, AuctionExplorer is a tool developed by Shmueli et al. [92] that allows collection, representation, and interactive visualization of auction databases. However, historical data on past auctions for a given item may not always be available, such as when the seller is introducing a new item to the market or if past data does not reflect the behavior of *current* bidders accurately.

Instead of using historical auction data to directly determine auction parameters, the data can instead be used to form bidding models. Bapna et al. [5] use inductive techniques to derive properties of bidder behaviors from auction data, resulting in a taxonomy of behaviors containing multiple strategies. They find considerable heterogeneity among auction participants and suggest that the induced behavior patterns should replace theory-driven models of bidders so as to facilitate the design of auction mechanisms. The resulting bidder models can be used for different purposes. By using these models in simulation, it is possible to explore the outcomes for different auction designs and gain intuition into setting auction parameters. For example, Bapna et al. [4] find that the bid increment has a large impact on bidder behavior in ascending multiunit auctions and offer a heuristic for choosing the increment. This use of bidder models in simulation to develop insights into the setting of auction parameters is similar to the use of simulated bidders in the metalearning step proposed in Section 3.2.1, but it is not clear what would happen if these bidder models are not representative of the encountered bidders.

By combining these modeled behaviors with Bayesian inference, it is possible to model bidders from limited experience in an online fashion, and then to use these models to set auction parameters. Bapna et al. [6] use this approach to classify bidder strategies and estimate their valuations in multiunit ascending auctions, and they are able to dynamically set bid increments such that auction revenue is higher than it would have been if a fixed, predetermined bid increment had been used. Rogers et al. [80] apply a similar approach to setting bid increments in repeated English auctions, using the closing prices of past auctions to estimate the number of bidders and their valuation distribution. These approaches are essentially the same as the Bayesian approach discussed in Section 3.2.2, and they therefore have the same drawback: if the encountered bidders do not conform to the assumptions of the models, then the auction parameters suggested by the models may be arbitrarily bad.

One interesting possibility that has not been considered in this dissertation is that of adapting parameters *during a single* auction. Bapna et al. [7] show how an understanding of how prices change during an auction can help improve prediction of auction revenue and the setting of parameters that may promote a favorable winning bid. By using regression to model the evolution of winning bids throughout an auction, they can predict how a change to a parameter such as the minimum bid increment at any point in time would affect the future price trajectory, and thus the closing price.

There has also been work exploring the adaptation of electronic auction mechanisms in response to bidder behavior from an empirical standpoint using a variety of learning approaches in simulated experiments. Cliff [27] explores a continuous space of auction mechanisms defined by a parameterized continuous double auction, where the parameter represents the probability that a seller will make an offer during any time slice. Bidders adjust their bids over time in response to observations of the market. The adaptive strategy of the bidders has 60 parameters that must be set. The mechanism parameter and the parameters of the simulated bidding agents used are learned through a coevolutionary process – a genetic algorithm is used to explore the parameter space, and the sets of parameters in each generation are evaluated by having agents with these parameter sets compete for profits in simulated auctions. For different underlying supply and demand schedules, the system converges to different values of the auction parameter. Phelps et al. [77] also address continuous double auctions, using genetic programming to co-evolve buyer and seller strategies and auction rules from scratch.

Byde [24] takes a similar approach in studying the space of auction mechanisms between the first and second-price sealed-bid auction. The winner's payment is determined as a weighted average of the two highest bids, with the weighting determined by the auction parameter. For a given population of bidders, the revenue-maximizing parameter is approximated by considering a number of parameter choices over the allowed range, using a genetic algorithm to learn the parameters of the bidders' strategies for each choice, and observing the resulting average revenues. For different bidder populations (factors considered include variable bidder counts, risk sensitivity, and correlation of signals), different auction parameter values are found to maximize revenue.

There are two primary difference between these three evolutionary approaches and the metalearning approach to developing adaptive auction mechanisms described in Section 3.2.1. First, these approaches learn auction parameters and bidder behaviors simultaneously, while the metalearning process makes use of a predetermined space of bidder populations that reflects our prior knowledge or experience with bidders. Second, at the end of the coevolutionary process, the final output of these approaches is a fixed mechanism. Although the auction mechanisms developed by these approaches may work well under the assumed conditions, when they are used in real-life settings the same problem may arise as with analytical mechanism design: bidders' goals, beliefs, and strategies may be different from those assumed, leading to unexpected results. In contrast, the goal of the work described here is to allow the auction mechanism to adapt in response to the encountered bidder population. While the means of adapting auction parameters used by these approaches in simulation could be applied in an online setting, they would likely be found unsuitable. For example, evolutionary methods frequently explore highly suboptimal solutions that could be disastrous if actually tried. Essentially, these methods do not address the exploration-exploitation tradeoff, focusing only on exploration. The methods used to evolve bidder strategies, however, could possibly be applied to generate the bidder behavior needed during the metalearning step described here.

One piece of related work that is not directly auction-related is that of Brooks et al. [21], who consider the problem of competing producers of information goods that must choose how to bundle and price these goods. The producers use a genetic algorithm to search the space of possible actions, and it is shown that profits are increased if knowledge of the market is used to either reduce the search space, start the search in a promising area, or introduce a gradient. This idea of using prior knowledge to direct the search for the optimal parameters of a sales strategy is similar to the use of metalearning to find the optimal learning parameters of the regression adaptive algorithm.

The lack of assumptions about bidders differentiates this work from Conitzer and Sandholm's *automated mechanism design* [29], in which mechanisms are computationally developed for specific situations based on complete knowledge of the distribution over bidders' types.

Loss aversion (exhibited by the bidders in the reserve price setting of Section 2.3.1) was first demonstrated by Kahneman and Tversky in their work on prospect theory [49], and there is empirical evidence that bidders in auctions are often loss averse [34] as well as experimental evidence that being loss averse can indeed affect the bidding strategy adopted by a bidder [35].

The case of bidders participating in a number of sequential auctions, addressed in the BIN price setting of Section 2.3.2, has received much recent attention, including work that offers insights into bidders behaviors [48, 113] or examines stylized sequential auctions to derive equilibrium bidding strategies analytically [44]. Interestingly, while Zeithammer [113] finds that data on eBay bidders suggests bidders are taking future auctions into account, Juda and Parkes [48] show that eBay bidders tend to bid inefficiently in sequential auctions. This irrationality may manifest itself differently in different bidder populations and auction settings; however, adaptive auction mechanisms are particularly designed to accommodate such uncertainty.

Reinforcement Learning [102] encompasses a wide variety of algorithms and has been applied in numerous domains. The particular case of a seller responding to customer behavior by using a k-armed bandit algorithm has been well studied (e.g. [83]), but generally in the context of price selection. One variant of k-armed bandits that is potentially related to our learning problem is that of contextual bandits [64], in which the distribution of rewards at each step is conditioned on a piece of information (representing context) that is observable by the agent before an action must be chosen. In our auction setting, context could represent slight differences between auctions that influence bidder behavior, such as time of day or a variation in the item being sold.

In general, the term metalearning describes a wide variety of approaches to improving the performance of a learning system for a particular task through experience with related tasks [18, 107]. Methods of achieving this goal include using meta-data (such as the characteristics of various tasks and the effectiveness of specific learning algorithms on each) to select the best learning algorithm to use, and making use of the knowledge gained on one previously seen task to improve learning on a related task (sometimes called *learning to learn* [104]; the methods of transfer learning described in Chapter 8 that make use of models learned on source data sets could be seen as fitting this category).

10.2 Supply Chain Management

I now turn to work related to the supply chain management scenario discussed in Chapters 4 and 5, beginning with general work, and then work on the specific subproblems an agent must handle.

Outside of TAC SCM, much of the work on agent-based supply chain management has focused on the design of architectures for distributed systems in which multiple agents throughout the supply chain must be able to communicate and coordinate. Fox et al. [39] discuss issues that arise when distributing activities among agents in a supply chain and present a software architecture and a set of agent components that permit coordination among agents. Two realistic supply chain applications implemented in this architecture, and experiments show how unexpected events that perturb the supply chain can be successfully handled. Sadeh et al. [86] developed MASCOT, an agent-based architecture for coordinated supply chain planning and scheduling. MASCOT allows users at different levels of the architecture to evaluate the effects that different actions will have across the supply chain. Aslam and Ng [2] present an agent-based environment for multi-objective supply chain optimization. A multi-level architecture allows simulations required for optimization to be carried out at different levels of fidelity throughout the supply chain, and intelligent optimization choices by agents can significantly reduce the size of the overall search space, greatly reducing computation time.

These systems typically involve a static supply chain, but it is also possible to allow for the dynamic formation of supply chains. Chen et al. [26] present a framework in which a virtual supply chain emerges from the decisions of independent agents to join or leave the system. A negotiation protocol is described that allows agents to form relationships with each other, and the resulting supply chain can be modeled using colored petri nets.

Within the TAC SCM scenario, we are interested mainly in the decisions of a single manufacturer operating in a fixed supply chain. A number of agent descriptions for TAC SCM have been published presenting various solutions to the problem.[1] Ketter et al. [53] present a survey of agent designs showing the variety of approaches that have been taken by competition entrants. At a high level, many of these agents are similar in design to TacTex – they divide the full problem into a number of smaller tasks and generally solve these tasks using decision theoretic approaches based on maximizing utility given estimates of various values and prices. The key differences are the specific methods used to solve these tasks.

10.2.1 Price Prediction

In work outside of TAC, the problem of predicting the probability of winning an auction with a particular sealed bid is commonly approached through statistical methods such as those surveyed by Papaioannou and Cassaigne [75]. A similar machine learning approach is developed by Lawrence [66] that uses a naive Bayes classifier to predict the probability of a bid winning based on the bid price, features of the RFQ, and available information about other bidders. Such methods often require extensive historical information about competitors' past bids and assume a static environment. In TAC SCM, probabilities vary considerably throughout the game, and very little information is available about competitors' bids while the game is running, hence the decision to use a particle filter (Section 5.1.1) in TacTex to model current-day bid distributions of computer prices.

As described in Section 5.3.2, DeepMaize (Kiekintveld et al.) [55] makes predictions for current and future computer prices using a k-nearest neighbors algorithm. To estimate current component prices, DeepMaize records the recent

[1] See `http://tradingagents.org` for a complete collection of papers on TAC SCM agents.

prices offered by each supplier and performs linear regression to estimate prices on each future due date. To model the changes in future component prices, DeepMaize uses a form of regression tree to learn the difference between actual observed prices in a data set of past games and the predictions of the linear interpolation method.

Prediction techniques that have been documented by other TAC SCM agents generally concern computer prices. A previous version of DeepMaize used equilibrium analysis to make predictions about the future state of the market, from which information such as future prices could be extracted [57]. CMieux (Benisch et al.) [13] makes predictions about computer prices using a form of modified regression tree called a distribution tree that learns to predict a distribution over winning prices using data from past games. For current component prices, CMieux predicts the price that will be offered for an RFQ with a given due date by using a nearest neighbors approach that considers recent offers with similar due dates. Ketter et al. [54] developed a method of modeling computer prices based on "economic regimes" representing specific underlying market conditions. The distribution of winning bids is modeled as a Gaussian mixture model, and by clustering these distributions, economic regimes can be observed that correlate with specific market conditions such as component scarcity. By modeling transitions between regimes, future price distributions can be predicted. Foreseer (Burke et al.) [23] uses a form of online learning to learn multipliers indicating the impact of various RFQ properties on current computer prices. An early version of Botticelli (Benisch et al.) [10] used a heuristic in which linear regression is performed on recent computer prices to predict a distribution over winning prices.

Although several agents make efforts to adapt to changing conditions *during* a single game, such as MinneTAC (Ketter et al.) [54] and SouthamptonSCM (He et al.) [42], methods of adaptation to a set of opponents over a series of games in TAC SCM (as explored in Chapter 9) have not been documented by any other agent.[2] Such adaptation has been used in the TAC Travel competition, however, both during a round of competition [98], and in response to hundreds of previous games [99].

10.2.2 Bidding on Customer RFQs

The problem of bidding on customer RFQs has been addressed with a wide variety of solutions. Southampton (He et al.) [43] takes a fuzzy reasoning approach in which a rule base is developed containing fuzzy rules that specify how to bid in various situations. PSUTAC (Sun et al.) [101] takes a similar knowledge-based approach. RedAgent (Keller et al.) [52] uses a simulated internal market to allocate resources and determine their values, identifying bid prices in the process. The approach used by TacTex, in which probabilities of offer acceptance are predicted and then used in an optimization routine, is also used in various forms by several other agents.

10.2.3 Production Scheduling

Like TacTex, many agents use some form of greedy production scheduling, but other, more sophisticated approaches have been studied. These include a stochastic programming approach used by Botticelli [12], in which expected profit is maximized through the use of samples generated from a probabilistic model of possible customer orders, and an approach treating the bidding and scheduling problems as a continuous knapsack problem [9]. In the latter case, an ϵ-optimal solution is presented which is shown to produce results similar to the greedy approach of TacTex but in significantly less time for large problems.

[2] Per personal communication, at one point DeepMaize did include data from past games in its k-nearest neighbors model.

10.2.4 Component Procurement

For component procurement, most agents now employ approaches that involve predictions of future component needs and prices and are somewhat similar to the approach used by TacTex. These approaches are often heuristic in nature, although there are some exceptions; Buffett and Scott [22] model the procurement problem as a Markov decision process and use dynamic programming to identify optimal actions.

Much of the attention that has been paid to the problem of component procurement has focused on the beginning of the game. In particular, an unintended feature of the initial game rules caused many agents to purchase the majority of their components at the very beginning of the game [56]. Although this issue was eliminated by rule changes in 2005, the correct approach to procurement on the first few game days remains an open question. During this period, information about supply and demand is limited, but prices tend to be fairly low. The current top agents (including TacTex and DeepMaize) follow the strategy introduced by PhantAgent (Stan et al.) [96], which is to place a particular pattern of moderately large orders of each component spaced throughout the game to ensure a minimum level of components at a reasonable price. Partly in response to this issue, in 2007 the TAC SCM Procurement Challenge [87] was held (alongside the Prediction Challenge described in Section 5.3). In this challenge, a modified version of the SCM server was used that allowed agents to negotiate long-term flexible-quantity procurement contracts at the beginning of the game in addition to the regular RFQ system.

10.3 Ad Auctions

I now discuss work related to ad auctions, the topic of Chapters 6 and 7. As in the previous section, I discuss both general work and work specific to TAC.

Keyword auctions have received a significant amount of attention in recent years. Much of this attention has focused on analyzing the properties of the generalized second price auction that is commonly used. For example, Edelman et al. [38] and Varian [105] show that in a full information setting where each bidder knows the valuations of other bidders, a continuum of Nash equilibria exists, including one that is socially optimal, but it is not clear what bidding strategies would allow bidders to reach an equilibrium.

Much of the work on bidding agents has focused on specific subproblems, rather than designing complete agents that solve bidding problems of the scope faced in TAC AA. One exception is an agent designed by Kitts and LeBlanc [58] to bid on a family of keywords in live Overture auctions. The agent bears a strong resemblance to TacTex at a high level and performs many of the same tasks, such as predicting the position and resulting profit for a given bid and optimizing bids over a time horizon. One key difference is that the agent does not have access to the underlying model of bidder behavior present in TAC AA and so must directly model the probabilities of events such as clicks and conversions using regression on recent data. In an experiment in live auctions, the agent was able to generate four times as many clicks as human managers for a reduced cost, demonstrating the value of research into bidding agents.

Among work on specific subproblems, common topics include keyword selection, click prediction, and bidding strategies. Rusmevichientong and Williamson [84] address the problem of choosing which subset of all possible keywords to bid on. Choosing a subset of keywords is not an important problem in AA where the set of keywords is small, but is very important in real auctions with thousands of keywords to choose from. They present an adaptive algorithm for experimenting with keywords for which clickthrough rates are unknown. In practice, this algorithm is able to converge to a nearly-optimal set of keywords quickly, even when the set of possible keywords is large. Richardson [79] describes a logistic regression model that can be used to predict click probabilities of new ads based on properties such as the length and text of the ad, the advertiser, and the page linked to. Zhou and Naroditskiy [114] address the problem of choosing bids across multiple auctions when costs per click, values per click, and clickthrough rates are known for each keyword and position and given a budget. They model the problem as a stochastic multiple-choice knapsack problem and present an algorithm that achieves nearly optimal performance on real bidding data sets.

Published reports of other TAC AA agents are still limited, but they nevertheless encompass a wide range of agent designs. QuakTAC (Vorobeychik) [108] simplifies the bidding problem by bidding some fraction α of its myopic value per click on each keyword. The parameter α is determined through simulation-based game theoretic analysis. DNAgent (Munsey et al.) [72] adjusts its daily bids according to a lookup table indexed by capacity, position, and query information. The table is evolved over many games using a genetic algorithm.

The agent most similar to TacTex is Schlemazl (Berg et al.) [14], one of the top agents from 2009 and 2010. Schlemazl uses regression algorithms from WEKA to build models of impression count, click probability, conversion probability, and cost per click. In contrast to TacTex, only the relationship between the bid amount and the quantity of interest is modeled (i.e., ad position is not taken into account). In addition, these models do not consider the actions of other bidders or underlying parameters such as user state. The optimization problem resulting from the outputs of these models is cast as a multiple choice knapsack problem and solved using an extension of the method of Zhou and Naroditskiy [114]. Berg et al. compare the effectiveness of this agent to simpler rule-based agents and find it to be more profitable; however, an agent that simply attempts to equate return on investment (in this case, profit per conversion, as capacity is the limiting factor) across all queries by revising bids each day is surprisingly effective. In an experiment in which an increasing amount of random noise is added to the models, the performance of Schlemazl quickly falls behind that of this simpler agent, showing that accurate estimation of the quantities modeled is important to agent performance and possibly a reason for the success of TacTex.

There is currently limited information about the approaches taken by the other TAC AA agents, but we are aware that they include decision-theoretic optimization similar to TacTex, rule-based systems, and genetic algorithms. Optimization-based approaches appeared to fare the best in general, and we believe that what sets TacTex apart in this area is its ability to estimate the full game state. This observation is consistent with past TAC scenarios (SCM and Travel). Successful agents tend to have similar optimization routines at their core, and the top agents are those that are best able to model their environment, with increasingly complex modeling approaches developed as the competitions mature.

10.4 Regression Transfer

Finally, I discuss work that is related to the general transfer learning problem addressed in Chapters 8 and 9.

The lowest common denominator of transfer learning methods is the leveraging of information from a source domain to speed up or otherwise improve learning in a different target domain. Transfer learning bears resemblance to classic case-based reasoning [60], especially in the need to reason about the similarity between tasks and instances.

More recently, transfer learning has been studied in a variety of different settings in addition to the classification and regression discussed here. For instance, Mihalkova et al. [71] show how transfer can be applied to Markov logic networks, a combination of Markov networks with first order logic that can be useful for modeling relational datasets. The most difficult aspect of learning a Markov logic network is determining the structure, or set of first-order clauses, and the proposed method attempts to transfer structure from source to target by finding similarities.

Transfer learning has also been applied to reinforcement learning. Here, the goal is generally to make use of policies or value functions learned on one original task to speed learning in a new task. The primary difficulty in most cases is that the new problem has a different state space, action space, observation format, or goal, requiring some sort of mapping between the two tasks that may be either chosen by hand [103] or generated automatically from task descriptions [69]. (These methods could in theory be applied to the sequential auction scenario of Chapter 2, which was framed as an RL problem, but this scenario presents different challenges. Here, the structure of the problem remains the same between tasks (the auction format and parameterization remain the same from market to market), but the behavior of the market participants causes the dynamics of the task to differ. Another difference is that experience from a space of possible bidder populations is used, rather than experience from only a single task.)

The most closely related work to this research, that on boosting and classification transfer, is described in Chapter 8. A key property of the classification and our regression setting is that the source and target domains typically have the same input and output spaces (X and Y in our notation), which is not always the case, for example in the reinforcement learning setting. As such, the problem studied here could be considered one of concept drift [89], in which the target concept changes over time. A related problem is covariate shift [91], in which the concept (Y given X) remains the same but the distribution over inputs (X) changes. In fact, the problem of training a model on one set of TAC games and applying it to another set of games with the *same* agents could be seen as one of covariate shift: the concept (determined by agent behavior and underlying game mechanics) remains the same, but the market conditions that arise in the particular games could differ between the sets.

Having identical input and output spaces between source and target also differentiates our setting from multitask learning, in which multiple related concepts sharing an input representation but with potentially unrelated outputs are to be learned simultaneously; however, some multitask learning methods could potentially be modified to address our setting. Caruana [25] showed that training a neural network with multiple outputs representing different tasks could lead to better performance on each task than training a separate network for each task individually. The obstacle to applying this approach to our transfer setting is that labels reflecting the source concept are not available for the target inputs, and vice-versa, but one potential solution would be to train source models, evaluate these models on the target inputs, then train a network on the target inputs with both the source and target labels as outputs.

Another area that is related to this work is ensemble learning in general, and expert prediction algorithms in particular. A number of online learning algorithms exist that combine advice from different experts, such as Littlestone and Warmuth's weighted majority algorithm [67]. While such algorithms typically only consider a fixed set of experts (i.e., no models trained on target data), their incremental nature could make them useful in situations where frequent retraining is not practical.

10.5 Summary

This chapter has discussed related work in several areas, including work on the market domains addressed in this dissertation (sequential auctions, supply chain management, and ad auctions), the specific agent tasks faced (including parameter tuning, bidding in various types of auctions, planning and scheduling, and modeling other agents) and the relevant learning paradigms (reinforcement learning, metalearning, supervised learning, and transfer learning). Only partial coverage of all related work has been possible given the wide variety of topics the work in this dissertation touches on, including the fields of electronic commerce, multiagent systems, machine learning, and economics. In fact, a key feature of this dissertation has been the integration of ideas from these areas into its central contribution: fully functional and demonstrably successful agents for realistic market domains that make effective use of machine learning, including the ability to incorporate prior market experience into the learning approaches used.

Chapter 11

Conclusion

This chapter concludes the dissertation by summarizing the work presented and discussing possible directions for future work.

11.1 Summary of Contributions

In this dissertation, I presented successful trading agents for three different market scenarios. In the first scenario (discussed in Chapter 2), I described an agent that can maximize its revenue when selling a large number of some item through sequential auctions. The agent must choose a mechanism to use in each auction by setting certain parameters that define the mechanism, such as reserve prices or buy-it-now prices. The agent operates by using an *adaptive auction mechanism* which can learn the optimal auction parameters through repeated interaction with a population of bidders.

The second scenario was TAC SCM, in which an agent acts as a computer manufacturer and must manage its supply chain by purchasing components, assembling computers, and selling the finished products to customers. In Chapter 4, I presented our TAC SCM agent TacTex. TacTex operates by making predictions about various quantities (the availability and prices of components, the nature of customer demand, and the computer prices that will result) and then optimizing its buying and selling decisions with respect to these predictions. TacTex has been a regular finalist in the annual TAC SCM competition, and was the winning agent in 2005, 2006, and 2010.

The third scenario was TAC AA, in which an agent runs an advertising campaign by bidding in keyword auctions. In Chapter 6, I presented the TacTex agent for TAC AA. As in TAC SCM, TacTex operates by making several predictions and estimates and then planning its future bids given this information. In this case, the quantities of interest include the numbers of users in different searching states and the bids of other advertisers. TacTex was the winner of the first two TAC AA competitions in 2009 and 2010.

In all three cases, the agents make use of machine learning. In the sequential auction scenario, the agent learns about the relationship between auction parameters and the resulting revenue using an adaptive algorithm. As the agent can only learn about the result of a particular parameter value by trying it, this is a reinforcement learning setting and presents an exploration-exploitation tradeoff. Experiments show that the regression adaptive algorithm of Chapter 2 is able to produce higher expected revenue than would be obtained by any fixed auction parameters. The TacTex agent for SCM learns models of future computer and component prices, and the TacTex agent for AA learns models of the bidding behavior of other advertisers. In the case of both TAC agents, the learning problem is one of supervised regression. The experiments of Chapters 5 and 7 show that improving the accuracy of these models can increase the profit earned by TacTex.

Although learning is important in all three scenarios, learning online from scratch can be slow. In many cases, an agent may have prior experience in other markets that are similar to the current market. The central question addressed in this dissertation was how to use this information:

How can adaptive trading agents take advantage of previous experience (real or simulated) in other markets, while remaining robust in the face of novel situations in a new market?

Chapters 3, 8, and 9 are devoted to answering this question by testing a number of different approaches under a variety of conditions, as outlined in Section 1.1. For each of the two learning settings considered (reinforcement learning and regression), one novel algorithm emerged that was effective across a wide range of experiments, as summarized in the remainder of this section.

In the sequential auction scenario, previous experience was represented by the ability to simulate a space of plausible bidder populations. As I showed in the experiments of Chapter 3, when the actual encountered bidder population was identical or similar to a population from this space, a Bayesian approach to identifying the population was able to learn effective auction parameters very quickly. However, if the encountered population exhibited unexpected behavior, the Bayesian approach could fail to find even reasonable auction parameters. A more robust solution to making use of simulated populations was the metalearning approach to setting the learning parameters of the regression adaptive algorithm by searching for the learning parameters that maximized revenue in simulation. The resulting learning parameters biased exploration in such a way that this approach was nearly as effective as the Bayesian approach when the encountered population resembled a simulated population, yet it maintained the ability to adapt when bidders behaved unexpectedly.

The supervised learning problem faced in both TAC scenarios can be framed as an instance of transfer learning. Here, the previous experience takes the form of data sets representing different markets (in SCM) or agents (in AA). This data is referred to as source data, and the goal is to make use of this data to learn a target concept for which only limited data is available. Previous work on transfer learning in supervised settings has largely focused on classification, and so in Chapter 8 I presented a comparison of algorithms for regression transfer, including modified classification algorithms and novel algorithms. These algorithms fall into two categories: those that use source data directly, and those that make use of models trained on the source data. I tested these algorithms across a wide range of

TAC and non-TAC data sets, and found that the algorithms that use source data directly tended to perform better. In particular, 2-Stage TrAdaBoost.R2 was consistently among the top algorithms on data sets for which boosting was effective. 2-Stage TrAdaBoost.R2 is a boosting algorithm that makes use of the basic principle introduced in the classification transfer algorithm TrAdaBoost [31] (reducing the weight of source instances that are poorly fit in each iteration), adapts this principle to regression using the weighting scheme used in AdaBoost.R2 [37], and adjusts the weights of source and target instances in two separate stages.

In Chapter 9, I tested some of the top regression transfer algorithms under a wide variety of conditions within the TAC scenarios. Again, 2-Stage TrAdaBoost.R2 was the top performer on data sets where boosting was effective. So long as the source and target were not identical, simply combining all source data and using 2-Stage TrAdaBoost.R2 produced excellent results regardless of the number of sources, similarity of source to target, or amount of source data. Transfer stacking, a method that combines source and target models, did not appear as robust, but it also performed well in certain cases, and both transfer stacking and 2-Stage TrAdaBoost.R2 were able to produce effective transfer when learning from completed TAC games as long as one-fold-per-game cross validation was used.

11.2 Future Work

A number of directions for future work exist for both the specific trading agents I have presented and the learning approaches they use.

11.2.1 Sequential Auctions

In the sequential auction scenario, several direct extensions to the algorithms presented are possible. One obvious need is to extend the regression adaptive algorithm to handle auctions with multiple parameters (for instance, both a BIN price and a reserve price). Locally weighted quadratic regression, the method used to estimate the auction outcome for a particular parameter value, can easily be extended to multiple dimensions. In fact, experiments not presented in this dissertation show that the regression adaptive algorithm can indeed be used to set multiple auction parameters. However, as the number of parameters increases, the rate of learning can decrease. One shortcoming of this direct extension to multiple parameters is that it assumes that the impact of each parameter is independent. It is likely the case that the relationship between parameters is more complex (for instance, in situations in which a high BIN price is optimal, a high reserve price might also make sense), and learning about this relationship from the simulated populations might be possible.

In the auction settings considered, the metalearning process was able to find effective learning parameters using SPSA, a gradient-based search method for stochastic approximation. In other settings, especially those with multiple auction parameters, a more sophisticated search method might be needed to avoid finding only a local maximum or simply to speed up the search. Another limitation of SPSA is that it requires the amount of prior experience (i.e., the number of initial data points used for regression) to be fixed in advance. An evolutionary search method that expanded the set of prior experience over time could possibly address both of these issues.

The Bayesian approach to identifying the bidder population can also be improved. This approach requires a representation of the distribution over auction outcomes given the bidder population and auction parameter value. In Chapter 3, this representation took the form of a table computed from many simulations using a discretized space of bidder populations, auction parameters, and outcomes. Even for the fairly coarse degree of granularity used and a bidder population space characterized by only four population parameters, the resulting table was quite large and time consuming to build. It might be possible to learn a model from a smaller amount of bidding data that is both more compact and more accurate.

Another improvement concerns the method of action selection used. In Chapter 3, softmax action selection was used to allow a direct comparison to the regression adaptive algorithm, but other possibilities exist that take into account the value of the information obtained by taking an action (e.g., [100]).

Given the success of the Bayesian approach when bidders behave as expected, and the robustness of the regression adaptive algorithm with metalearned parameters, an appealing option would be to combine these two methods. One potential solution is to start with the Bayesian approach, attempt to detect when the encountered bidder population does not match any of the populations from simulation, and then switch to the regression adaptive approach if that is the case. An alternative would be to choose at each auction between taking the action suggested by each algorithm by using an expert algorithm.

A more fundamental change to the algorithms presented in this dissertation would be to view the problem as one of multi-state RL instead of single-state RL. When bidders can remain across multiple auctions, as in the BIN setting described in Section 2.3.2, then the auction parameter chosen will impact not only the auction outcome, but also the set of bidders participating in the next auction. (For example, setting a low BIN price might result in the current auction ending sooner and more bidders remaining for the following auction.) In this situation, we would need to model both auction outcomes and state transitions, where a state represents the state of all bidders who will participate in an auction.

Finally, at a more general level, the work presented here on learning auction parameters is empirical in nature and focuses on maximizing expected revenue. Much theoretical work on sequential auctions in particular and online learning in general focuses on placing a bound on worst-case performance or regret. It might be worthwhile to address this problem from both directions to see if algorithms can be designed that perform as well on average as those presented here while still offering performance guarantees.

11.2.2 TAC Agents

One aspect of the TacTex agent for SCM that offers room for improvement is its long term planning, particularly in the area of component procurement. TacTex plans its short term orders (up to 10 days) based on the projected customer demand and its production schedule, but like many agents, it turns to more heuristic methods when placing orders further into the future, especially at the very beginning of the game.

Another issue is the fact that when learning models, TacTex is learning to make predictions about a market that includes itself. By updating its models, TacTex causes changes in its own behavior that can impact the prices it is trying to predict. To be more precise, if the training data used to learn model B covers games in which TacTex used model A, then when making decisions based on model B, TacTex should account for the difference between the impact on prices of the actions it plans based on model B and the actions it *would have* taken based on model A. Another interesting possibility when training data is generated from game simulations using agent binaries (as opposed to live competition data) would be to alternate playing games using the latest model and training on the latest batch of games until the models converged (if at all).

The TacTex agent for AA offers even more opportunities for improvement. One limitation of the current agent is that it performs optimization with respect to the *expected* actions of other bidders. Optimizing with respect to a set of sampled actions could produce better results, although it would be more time consuming.

There is also significant room for improvement in bid prediction. The results of Chapter 7 show the importance of using accurate bidder models, and there are a number of additional conditional density estimation approaches we could try. One additional consideration is running time. The particle filter described in Section 7.1 can be fairly time consuming, which is a problem in both TAC and real world auctions, and the most costly step is repeatedly calling the bidder models to generate the distributions over the next bids. Reducing the time needed to evaluate these models would be an important part of scaling up the proposed approach to handle thousands of real auctions with many bidders. Another limitation is that we currently only estimate the bids for the previous day, then assume that these bids persist. The next step is to predict future bids, perhaps by using the bidder models to propagate the estimates forward.

Also, as mentioned in the experiments of Chapter 7, the problem of estimating bids is closely tied to the problem of estimating spending limits. Rather than estimating the two independently, as TacTex currently does, it might be possible to incorporate spending limit estimation into the particle filter for bid estimation. The key challenge to such an approach would likely come in the machine learning component, as we would now need to generate a distribution over the next day's (bid, spending limit) pair, instead of just the bid.

The approach taken by both TacTex agents is essentially decision-theoretic: the behavior of other agents in the market is assumed to be fixed, and TacTex then optimizes with respect to this expected behavior. Because other agents have the opportunity to respond to the actions of TacTex, game-theoretic considerations should also be taken into account. TAC AA lends itself to such analysis particularly well, as agents can directly observe and respond to the actions of others (for example, outbidding another agent might cause that agent to return the favor on the following day or move to a different auction altogether).

Finally, an important step in verifying the general applicability of the algorithms developed for both TacTex agents is applying them to real world market situations. Certain components of these agents are more directly transferable to real markets than others. Many of the underlying models used by these agents are specific to the TAC specification, such as the supplier model for TAC SCM (Section 4.5.1) and the user model for TAC AA (Section 6.4). In real markets, models specifying the behavior of market participants are not provided, and so developing these models would be an added challenge for an agent developer. On the other hand, some of the methods of using these models for prediction and optimization could potentially be of use to agents in real markets. In particular, the particle filter for bid estimation in keyword auctions described in Section 7.1 is applicable under fairly general conditions, and is thus a top candidate for real-world experiments. In addition, it is hoped that the general learning methods used (including the transfer learning algorithms developed) would be widely applicable to market prediction problems.

One complication in applying the learning approaches used to real world data is the fact that fully detailed historical data may not be available in many settings. For the TAC scenarios, I have assumed that the game logs, which contain all prices and bids of interest, are available to the agent to use as training data. In some cases (such as SCM computer prices), this information could be estimated with reasonable accuracy from the agent's own observations. In other cases (such as the bids of others in AA), this information would be very difficult to obtain. One question that would thus need to be addressed is how accurate historical data must be to be of use to an agent. In some cases, it might be possible to use the agent's own modeling abilities to improve the accuracy of the available data. For instance, in keyword auctions, it might be possible to form initial bid estimates using the limited information released by a search engine, then bootstrap by alternating model building and particle filtering stages to obtain increasingly accurate estimates.

11.2.3 Regression Transfer

Chapter 8 compared several algorithms for regression transfer, but there are additional algorithms that could also be tried. As was mentioned, the algorithms chosen were those that can be used in conjunction with any regression base learner.

Algorithms that were ruled out due to this constraint should also be evaluated where possible. Several of the algorithms presented, including 2-Stage TrAdaBoost.R2, are based on boosting and make use of the weighting scheme of AdaBoost.R2. Additional methods of adapting boosting for regression could be explored, and additional techniques for improving boosting (such as regularization) could be tried.

In some cases, the algorithms of Chapter 8 were inspired by or directly based on algorithms for classification transfer. It would also be interesting to see if adapting algorithms in the other direction is useful. In particular, some of the modifications made to TrAdaBoost (primarily the two stage approach) could be directly applied in a classification setting, and it would be worthwhile to explore whether improvement in that setting is also observed.

The experiments of Chapters 8 and 9 considered cases where a fairly small number of source data sets were available. The effectiveness of the regression transfer algorithms when many sources are available remains an open question, and perhaps a means of selecting which sources should be used could be developed.

Finally, one important item that has not been discussed, as Chapters 8 and 9 were empirical in focus, is the theoretical properties of the algorithms presented. One of the attractive features of AdaBoost is its theoretical guarantees (e.g., convergence to zero error on the training set [40]). Analogues of the main properties of AdaBoost have been proven to apply to TrAdaBoost [31], and it can be shown that these properties hold in modified form when TrAdaBoost is modified for regression using one particular weighting scheme (AdaBoost-Δ [15]). However, the AdaBoost.R2 weighting scheme that proved most effective in the experiments of this dissertation is more difficult to analyze, and the addition of the two-stage procedure used in 2-Stage TrAdaBoost.R2 further complicates matters. Developing algorithms that are both effective in practice and offer theoretical guarantees is an important area for future work.

11.3 Conclusion

The growth of electronic commerce continues at a rapid pace, and in future years, new markets and new opportunities for trading electronically will certainly appear. Intelligent agents that can rapidly make a large volume of trading decisions with little or no human supervision will play a growing role in these marketplaces, and research into developing such agents will continue.

In the Trading Agent Competition, progress over the years within each scenario (such as SCM or AA) has followed a similar trajectory. Simple agents are sometimes successful in early years, but over time, the top agents become increasingly complex, and an emphasis is often placed on applying machine learning to making more accurate predictions or modeling other agents. A similar pattern can occur when developing agents to participate in real markets. Once the low-hanging fruit is gone, maintaining a competitive edge requires an increasing degree of sophistication.

One advantage an agent can have is the ability to adapt rapidly to new situations. Machine learning can be a useful tool for adaptation, but learning requires experience. In this dissertation, I addressed the question of how an agent can quickly learn about a new market by making use of experience in other markets. I introduced new learning algorithms that were effective under a variety of conditions within specific market scenarios, and I believe these algorithms should generalize well to other domains. It is my hope that this work will serve as a useful resource for agent developers and provide inspiration for continued research on adaptive trading agents.

Appendix

Appendix 1

Position Analyzer for Ad Auctions

This appendix describes the Position Analyzer from the TacTex agent for Ad Auctions, as referenced in Section 6.3.

For each query type, the position analyzer takes as input i) the average position of each advertiser, ii) the number of impressions for TacTex, and iii) an upper bound on the total number of impressions. Using this information, the position analyzer attempts to determine i) the ranking of the squashed bids, and ii) the first and last impression for each advertiser – in other words, it attempts to reconstruct the impression range column from Table 6.1 (reprinted here as Table 1.1).

For advertiser a, the three values that we wish to determine are i) $rank_a$, the ranking of the advertiser's squashed bid (1-8); ii) $start_a$, the first impression for which the advertiser's ad was displayed; and iii) end_a, the impression after the last. As part of the process we will also need to determine $limitOrder_a$, the order in which the advertiser hit its spending limit with respect to other advertisers (again 1-8, with ties broken by rank). If $rank_a < 6$, $start_a$ will be 1; otherwise, it will be end_b, where b is the advertiser for which $limitOrder_b = rank_a - 5$ (the fact that b hits its spending limit is what causes a to rise into fifth position). If advertiser a never hits its spending limit, then end_a will be the total number of impressions plus one. Table 1.2 shows all of these values for the query shown in Table 6.1.

The game server computes the average position for advertiser a by taking the sum of positions over all of the advertiser's impressions (sum_a) and dividing

	Agent actions				*Results*					
Advertiser	*Bid*	*Sq bid*	*Ad*	*Sp lim*	*CPC*	*Imps*	*Clks*	*Cnvs*	*Impression range*	*Avg pos*
MetroClick	0.315	0.109	generic	50.93	0.310	426	164	16		1.000
QuakTAC	0.266	0.107	lioneer:dvd	-	0.194	718	156	6		1.593
TacTex	0.235	0.091	generic	0.236	0.201	77	1	0		3.000
UMTac09	0.216	0.078	generic	7.583	0.209	700	36	6		2.719
munsey	0.190	0.075	generic	-	0.174	718	16	2		3.675
epflagent	0.214	0.068	generic	-	0.184	641	3	0		4.510
AstonTAC	0.158	0.059	generic	500.0	0.133	292	1	0		4.938
Schlemazl	0.062	0.020	flat:dvd	5.617	-	0	0	0		-

Table 1.1: Results for the query *null:dvd* from one game day of the 2009 TAC AA finals

by the number of impressions:

$$averagePosition_a = \frac{sum_a}{end_a - start_a} \tag{1.1}$$

Suppose that we already know each *rank* value. Then we can express sum_a as follows:

$$sum_a = \sum_{b\,:\,rank_b \leq rank_a} max(0, min(end_b, end_a) - start_a) \tag{1.2}$$

The contribution of each advertiser b to sum_a is the number of impressions for which its ad was displayed above the ad of advertiser a (advertiser a's ads are included because the ranking is 1-based). If we also know each *limitOrder* value, then we can rewrite sum_a without the *max* and *min*, giving us a system of linear equations that we could try to solve. However, we have twice as many variables as equations ($start_a$ and end_a for each advertiser a), giving us a severely under-constrained system. In addition, we do not in fact know the *rank* and *limitOrder* values, and would need to try solving the system corresponding to each set of values.

An additional piece of information that we have not taken advantage of is the fact that despite being represented by a Java double, each average position is a fraction (as shown in Equation 1.1). We can therefore use the method of continued fractions to find the fractional representation for each average, $numerator_a/$

Advertiser	*Rank*	*Start*	*End*	*Impression range*	*LimitOrder*	*Avg pos*	*Fraction*	*GCD*
MetroClick	1	1	427		2	1.000	1/1	426
QuakTAC	2	1	719		4	1.593	572/359	2
TacTex	3	1	78		1	3.000	3/1	77
UMTac09	4	1	701		3	2.719	1903/700	1
munsey	5	1	719		5	3.675	2639/718	1
epflagent	6	78	719		6	4.510	2891/641	1
AstonTAC	7	427	719		7	4.938	721/146	2
Schlemazl	8	-	-		-	-	-	-

Table 1.2: Values computed by the Position Analyzer for the data from Table 1.1

$denominator_a$. These values are shown in Table 1.2. (Note that the average positions shown are truncated.) In some cases, the numerator and denominator of this fraction will equal the values shown in Equation 1.1, giving us exactly the information we need. For example, using average position for UMTac09, we obtain a fraction of 1903/700, and UMTac09 indeed had 700 impressions. However, if the fraction in Equation 1.1 is reducible, with the numerator and denominator having a GCD of gcd_a, then we need to multiply the resulting numerator and denominator by the unknown value gcd_a to obtain the needed information. For example, for QuakTAC we obtain the fraction 572/359 and must multiply by 2, and for TacTex we obtain the fraction 3/1 and must multiply by 77. So now instead of two variables for each advertiser a, we have only one, gcd_a. We could again try to solve for these variables by setting up a system of linear equations, but in this case the system is homogeneous, again resulting in a large space of possible solutions. As before, we also do not know the *rank* and *limitOrder* values.

Instead of solving a system of equations, we perform a depth-first search of the space of possible values of each *rank*, *limitOrder*, and *gcd* by cycling through nodes of corresponding types in the search tree. Although the search space is large, by utilizing a number of pruning rules we can usually search the tree quickly. Figure 1.1 shows a portion of the search tree for the data from Table 6.1. For clarity, I will say that the *level* of a node is the number of ancestor nodes of the same type plus 1 (rather than its actual depth), and the nodes of Figure 1.1 are labeled with these levels. Algorithms 9-12 summarize the search process. Note that in these algorithms, all values to be determined are treated as global variables, and any variable set at a search node must be unset when backtracking.

The root of the tree is the level 1 rank node. At a level i rank node, we choose which advertiser a_i has rank i. An advertiser with average position p cannot have a rank less than $\lceil p \rceil$. Thus in Figure 1.1, there is only one choice at both $rank(1)$ and $rank(2)$, while at $rank(3)$ there are two choices.

Search then proceeds from a level i rank node to a limitOrder node of the same level. At this node we choose the order in which the advertiser a_i chosen at the preceding *rank* node hits its spending limit with respect to all previously chosen advertisers, set its *limitOrder* accordingly, and increment any *limitOrder* values that have already been set and are as large or larger. Again, our choices are constrained by the advertiser's average position. In order to have average position p, an advertiser's *limitOrder* cannot be less than $i+1-\lfloor p \rfloor$. (The agent starts in ith place and must move up to at least position $\lfloor p \rfloor$.) In Figure 1.1, at $limitOrder(1)$ we assign MetroClick a *limitOrder* value of 1 (the only possible value at level 1), at $limitOrder(2)$ we assign QuakTAC a *limitOrder* of 2 (it cannot be 1 according to the constraint), and at $limitOrder(3)$ we can assign UMTac09 a *limitOrder* of 2 or 3.

A level i limitOrder node leads to a level i gcd node. Here we must choose the value of *gcd* for the advertiser a_i chosen at the preceding *rank* node. Since we know the *rank* and *limitOrder* values of all advertisers with higher rank, and we know that the numerator and denominator of the fraction in Equation 1.1 are equal to $gcd_{a_i} numerator_{a_i}$ and $gcd_{a_i} denominator_{a_i}$, respectively, we can derive the following from Equations 1 and 2:

$$gcd_{a_i}(numerator_{a_i} - denominator_{a_i}(i - limitOrder_{a_i} + 1)) \;=$$

$$\sum_{\substack{b\,:\,rank_b<rank_{a_i},\\ limitOrder_b<limitOrder_{a_i}}} max(0, end_b - start_{a_i}) \tag{1.3}$$

If we know the *end* values for all advertisers with higher rank and both sides of the above equation are nonzero, then it is straightforward to determine $start_{a_i}$, use the equation to find gcd_{a_i}, and then compute end_{a_i}. Unfortunately, both sides of the equation will be zero whenever $limitOrder_{a_i} = 1$.

As a result, we have three cases to consider. In the first case, $limitOrder_{a_i} = 1$, and we continue without determining a value for end_{a_i} or gcd_{a_i}. In the second case, the right hand side of Equation 1.3 is nonzero and contains no unknown end_b values. In this case, we can solve for gcd_{a_i}. In the third case, the right hand side of Equation 1.3 is nonzero but does contain unknown values. In this case, we attempt to guess the value of gcd_{a_i}, and then we determine the values of $start_{a_i}$ and end_{a_i} (possibly obtaining expressions including these unknown end_b values). We also plug these values into Equation 1.3 and store the resulting equation. As we proceed down the search tree, we will usually obtain enough equations to be able to solve for any unknowns. If at any point the system of equations is inconsistent, we backtrack. The possible values of gcd_{a_i} are constrained by the *end* values of the advertisers with the previous and next values of *limitOrder*, and by the upper bound on total impressions. Still, a wide range of values may be possible. Because we have a limited amount of time to search, and because lower values of gcd_{a_i} are most common, we implement a form of iterative broadening search. We repeat the search until we find a solution or hit a time limit of 300ms, and on our nth attempt, we consider only the first $5n$ possible values for gcd_{a_i} at each gcd node. In some cases, there will be no possible values for gcd_{a_i}, and we backtrack. Finally, note that when advertiser a_i is our own agent, we know our own number of impressions and thus our *gcd*. Unless forced to backtrack, a level i gcd node leads to a level $i + 1$ rank node.

Figure 1.1 shows examples of cases 1 ($gcd(1)$) and 3 ($gcd(2)$ and $gcd(3)$). At

$gcd(1)$, we know $start_{a_1} = 1$ (because MetroClick is ranked in the top five), but we cannot determine gcd_{a_1} or end_{a_1}. At $gcd(2)$, we can also set $start_{a_2} = 1$. Because $limitOrder_{a_2} > limitOrder_{a_1}$, we know that $end_{a_2} \geq end_{a_1}$; however, we do not know end_{a_1}. As there are not yet any advertisers with a higher $limitOrder$, our only upper bound on end_{a_2} is the upper bound on the total number of impressions. Assuming this value is sufficiently high, we will have at least five values to consider for gcd_{a_2}. This is our first attempt at finding a solution, so we try only the first five (1 through 5). First, we try $gcd_{a_2} = 1$. Plugging known values into Equation 3 gives us $1 \cdot 213 = end_{a_1} - 1$, and so we can set $end_{a_1} = 214$. Note that this value is incorrect; it would be correct had we chosen the correct gcd_{a_2} of 2. We can also set $end_{a_2} = start_{a_2} + denominator_{a_2} gcd_{a_2} = 360$. At $gcd(3)$, we are forced to backtrack. Having chosen $limitOrder_{a_3}$ to be 2 (meaning $limitOrder_{a_2}$, for QuakTAC, has been incremented to 3) UMTac09 must hit its spending limit before QuakTAC, and so we must have $end_{a_3} < end_{a_2}$. But this implies $1+700gcd_{a_3} < 360$, an impossibility.

If we reach a rank node with all advertisers having been assigned a rank, then we have found a valid solution. If any variables remain unsolved, we set them to the median of their possible range. Because there may be multiple valid solutions, we record this solution and then backtrack to search for more. At the end of our search, if there is only one solution, we return it. In rare cases (around 1% of the time), we hit the 300ms time limit without finding any solutions. Often, multiple solutions are found, and we choose the best one by scoring each solution according to a number of heuristics. Solutions are favored if: i) the total number of impressions is low, ii) there are no unsolved variables, iii) multiple advertisers share the maximum *end* value, iv) no advertisers share any other *end* value, and v) only the advertiser with greatest *rank* has an average position of 5.0. Typically the solution chosen is correct or very nearly so.

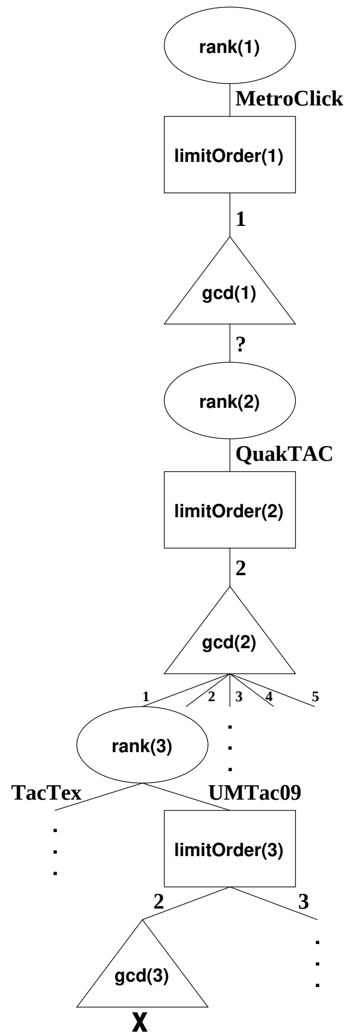

Figure 1.1: Part of the search tree for Table 1.1

Algorithm 9: Position-Analyzer

```
begin
    input  : all average positions, TexTex's impressions, upper bound
             on impressions
    output: ranking, impressions per advertiser, total impressions
    n ← 0
    Equations ← {}
    Solutions ← {}
    while Solutions = {}, time ¡ 300ms do
        n ← n + 1
        Rank-Node(1)
    Score solutions using heuristics and select the best
```

Algorithm 10: Rank-Node

```
begin
    input: level
    if level > number of advertisers with a position then
        valid solution found; add it to Solutions
    else
        determine set A of remaining advertisers that could    have rank
        level
        for ∀adv ∈ A do
            a_level ← adv
            rank_{a_level} ← level
            Limit-Order-Node(level)
```

Algorithm 11: Limit-Order-Node

```
begin
    input: level
    determine possible values of limitOrder_{a_level}
    for each value v do
        for 1 ≤ i ≤ level − 1 do
            if limitOrder_{a_i} ≥ v then
                limitOrder_{a_i} ← limitOrder_{a_i} + 1
        limitOrder_{a_level} ← v
        GCD-Node(level)
```

Algorithm 12: GCD-NODE

```
begin
    input: level
    if rank_{a_level} < 6 then
        start_{a_level} ← 1
    else
        find b such that limitOrder_b = rank_{a_level} − 5
        start_{a_level} ← end_b                    /* may include unknown */
    if limitOrder_{a_level} = 1 then
        leave gcd_{a_level} and end_{a_level} unset for now
        RANK-NODE(level + 1)
    else if RHS of Equation 1.3 has no unknowns then
        solve for gcd_{a_level}
        end_{a_level} ← start_{a_level} + denominator_{a_level} gcd_{a_level}
        if end_{a_level} is feasible then
            RANK-NODE(level + 1)
    else
        determine feasible values of gcd_{a_level}
        for up to 5n values v do
            gcd_{a_level} ← v
            end_{a_level} ← start_{a_level} + denominator_{a_level} gcd_{a_level}
            add Equation 1.3 to Equations
            try to solve for unknowns in Equations
            if end_{a_level} is feasible and Equations is consistent then
                RANK-NODE(level + 1)
```

Bibliography

[1] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[2] Tehseen Aslam and Amos Ng. Agent-based simulation and simulation-based optimisation for supply chain management. In Lihui Wang and S.C. Lenny Koh, editors, *Enterprise Networks and Logistics for Agile Manufacturing*. Springer, 2010.

[3] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[4] Ravi Bapna, Paulo Goes, and Alok Gupta. Analysis and design of business-to-consumer online auctions. *Management Science*, 49(1):85–101, 2003.

[5] Ravi Bapna, Paulo Goes, Alok Gupta, and Yiwei Jin. User heterogeneity and its impact on electronic auction market design: An empirical exploration. *MIS Quarterly*, 28(1):21–43, 2004.

[6] Ravi Bapna, Paulo Goes, Alok Gupta, and Gilbert Karuga. Predicting bidders willingness to pay in online multiunit ascending auctions: Analytical and empirical insights. *INFORMS Journal on Computing*, 20(3):345–355, 2008.

[7] Ravi Bapna, Wolfgang Jank, and Galit Shmueli. Price formation and its dynamics in online auctions. *Decision Support Systems*, 44(3):641–656, 2008.

[8] Michael Benisch, James Andrews, David Bangerter, Timothy Kirchner, Benjamin Tsai, and Norman Sadeh. CMieux analysis and instrumentation toolkit for TAC SCM. Technical Report CMU-ISRI-05-127, School of Computer Science, Carnegie Mellon University, September 2005.

[9] Michael Benisch, James Andrews, and Norman Sadeh. Pricing for customers with probabilistic valuations as a continuous knapsack problem. In *Eighth International Conference on Electronic Commerce*, pages 38–46, 2006.

[10] Michael Benisch, Amy Greenwald, Ioanna Grypari, Roger Lederman, Victor Naroditskiy, and Michael Tschantz. Botticelli: A supply chain management agent. In *Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 3, pages 1174–1181, 2004.

[11] Michael Benisch, Amy Greenwald, Ioanna Grypari, Roger Lederman, Victor Naroditskiy, and Michael Tschantz. Botticelli: A supply chain management agent designed to optimize under uncertainty. *SIGecom Exchanges*, 4(3):29–37, February 2004.

[12] Michael Benisch, Amy Greenwald, Victor Naroditskiy, and Michael Tschantz. A stochastic programming approach to scheduling in TAC SCM. In *Fifth ACM Conference on Electronic Commerce*, pages 152–159, 2004.

[13] Michael Benisch, Alberto Sardinha, James Andrews, and Norman Sadeh. CMieux: Adaptive strategies for competitive supply chain trading. In *Eighth International Conference on Electronic Commerce*, pages 47–58, 2006.

[14] Jordan Berg, Amy Greenwald, Victor Naroditskiy, and Eric Sodomka. A first approach to autonomous bidding in ad auctions. In *EC 2010 Workshop on Trading Agent Design and Analysis (TADA)*, Cambridge, Massachusetts, 2010.

[15] Alberto Bertoni, Paola Campadelli, and M. Parodi. A boosting algorithm for regression. In *ICANN '97: Proceedings of the 7th International Conference on Artificial Neural Networks*, pages 343–348, London, UK, 1997. Springer-Verlag.

[16] Avrim Blum and Jason D. Hartline. Near-optimal online auctions. In *SODA '05: Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1156–1163, 2005.

[17] Avrim Blum, Vijay Kumar, Atri Rudra, and Felix Wu. Online learning in online auctions. In *SODA '03: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.

[18] Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and R. Vilalta. *Metalearning: Applications to Data Mining*. Cognitive Technologies. Springer, January 2009.

[19] Leo Breiman. Bagging predictors. *Machine Learning*, 26:123–140, 1996.

[20] Leo Breiman. Stacked regressions. *Machine Learning*, 24:49–64, 1996.

[21] Christopher H. Brooks, Robert S. Gazzale, Jeffrey K. MacKie Mason, and Edmund H. Durfee. Improving learning performance by applying economic knowledge. In *Agent-Mediated Electronic Commerce V, Designing Mechanisms and Systems, AAMAS 2003 Workshop, Revised Selected Papers*, volume 3048 of *Lecture Notes in Computer Science*. Springer, 2004.

[22] Scott Buffett and Nathan Scott. An algorithm for procurement in supply chain management. In *AAMAS 2004 Workshop on Trading Agent Design and Analysis*, 2004.

[23] David A. Burke, Kenneth N. Brown, Brahim Hnich, and Armagan Tarim. Learning market prices for a real-time supply chain management trading

agent. In *AAMAS 2006 Workshop on Trading Agent Design and Analysis / Agent Mediated Electronic Commerce*, 2006.

[24] Andrew Byde. Applying evolutionary game theory to auction mechanism design. In *Proceedings of the 4th ACM Conference on Electronic Commerce*, pages 192–193, 2003.

[25] Rich Caruana. Multitask learning. In *Machine Learning*, volume 28, pages 41–75, 1997.

[26] Ye Chen, Yun Peng, Tim Finin, Yannis Labrou, and Scott Cost. A negotiation-based multi-agent system for supply chain management. *In Workshop on Agent-Based Decision Support in Managing the Internet-Enabled Supply-Chain, at Agents '99*, 1999.

[27] Dave Cliff. ZIP60: Further explorations in the evolutionary design of trader agents and online auction-market mechanisms. *Evolutionary Computation*, 13(1):3–18, 2009.

[28] John Collins, Raghu Arunachalam, Norman Sadeh, Joakim Eriksson, Niclas Finne, and Sverker Janson. The Supply Chain Management game for the 2006 Trading Agent Competition. Technical report, 2005. Available from http://www.sics.se/tac/tac06scmspec_v16.pdf.

[29] Vincent Conitzer and Tuomas Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, pages 132–141, 2004.

[30] Peter C. Cramton. The FCC spectrum auctions: An early assessment. *Journal of Economics and Management Strategy*, 6(3):431–495, 1997.

[31] Wenyuan Dai, Qiang Yang, Gui-rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the Twenty Fourth International Conference on Machine Learning*, pages 193–200, 2007.

[32] Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[33] Hal Daumé III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.

[34] Stefano DellaVigna. Psychology and economics: Evidence from the field. *Journal of Economic Literature*, 47(2):315–72, 2009.

[35] Dennis A. V. Dittrich, Werner Gth, Martin Kocher, and Paul Pezanis-Christou. Loss aversion and learning to bid. Papers on Strategic Interaction 2005-03, Max Planck Institute of Economics, Strategic Interaction Group, 2005.

[36] A. Dodonova and Y. Khoroshilov. Optimal auction design when bidders are loss averse. Working Paper. University of Ottawa.

[37] Harris Drucker. Improving regressors using boosting techniques. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 107–115, 1997.

[38] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97:242–259, 2007.

[39] Mark S. Fox, Mihai Barbuceanu, and Rune Teigen. Agent-oriented supply-chain management. *International Journal of Flexible Manufacturing Systems*, 12:165–188, 2000.

[40] Yoav Freund and Robert Schapire. A decision-theoretic generalization of online learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[41] Jerome Friedman. Multivariate adaptive regression splines (with discussion). *Annals of Statistics*, 19:1–141, 1991.

[42] Minghua He, Alex Rogers, Esther David, and Nicholas R. Jennings. Designing and evaluating an adaptive trading agent for supply chain management applications. In *IJCAI 2005 Workshop on Trading Agent Design and Analysis*, 2005.

[43] Minghua He, Alex Rogers, Xudong Luo, and Nicholas R. Jennings. Designing a successful trading agent for supply chain management. In *Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1159–1166, 2006.

[44] Thomas Jeitschko. Learning in sequential auctions. *Southern Economic Journal*, 65(1):98–112, 1998.

[45] Patrick Jordan, Ben Cassell, Lee Callender, and Michael Wellman. The Ad Auctions Game for the 2009 Trading Agent Competition. Technical report, 2009. Available from http://aa.tradingagents.org.

[46] Patrick Jordan, Akshat Kaul, and Michael Wellman. The Ad Auctions Game for the 2010 Trading Agent Competition. Technical report, 2010. Available from http://aa.tradingagents.org.

[47] Patrick Jordan, Michael Wellman, and Guha Balakrishnan. Strategy and mechanism lessons from the first Ad Auctions Trading Agent Competition. In *ACM Conference on Electronic Commerce*, pages 287–296, 2010.

[48] Adam Juda and David Parkes. An options-based solution to the sequential auction problem. *Artificial Intelligence*, 173:876–899, May 2009.

[49] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–291, 1979.

[50] Daniel Kahneman and Amos Tversky, editors. *Choices, Values and Frames.* Cambridge University Press, Cambridge, 2000.

[51] Toshihiro Kamishima, Masahiro Hamasaki, and Shotaro Akaho. TrBagg: A simple transfer learning method and its application to personalization in collaborative tagging. In *Proceedings of The 9th IEEE International Conference on Data Mining*, pages 219–228, 2009.

[52] Phillipp W. Keller, Felix-Olivier Duguay, and Doina Precup. RedAgent - winner of TAC SCM 2003. *SIGecom Exchanges: Special Issue on Trading Agent Design and Analysis*, 4(3):1–8, Winter 2004.

[53] Wolfgang Ketter, John Collins, and Maria Gini. A Survey of Agent Designs for TAC SCM. In *AAAI Workshop on Trading Agent Design and Analysis*, Chicago, USA, July 2008.

[54] Wolfgang Ketter, John Collins, Maria Gini, Alok Gupta, and Paul Schrater. Identifying and forecasting economic regimes in TAC SCM. In *IJCAI 2005 Workshop on Trading Agent Design and Analysis*, pages 53–60, August 2005.

[55] Christopher Kiekintveld, Jason Miller, Patrick R. Jordan, Lee F. Callender, and Michael P. Wellman. Forecasting market prices in a supply chain game. *Electronic Commerce Research Applications*, 8(2):63–77, 2009.

[56] Christopher Kiekintveld, Yevgeniy Vorobeychik, and Michael P. Wellman. An analysis of the 2004 supply chain management Trading Agent Compe-

tition. In *IJCAI 2005 Workshop on Trading Agent Design and Analysis*, 2005.

[57] Christopher Kiekintveld, Michael Wellman, Satinder Singh, Joshua Estelle, Yevgeniy Vorobeychik, Vishal Soni, and Matthew Rudary. Distributed feedback control for decision making on supply chains. In *Fourteenth International Conference on Automated Planning and Scheduling*, pages 384–392, 2004.

[58] Brendan Kitts and Benjamin Leblanc. Optimal bidding on keyword auctions. *Electronic Markets*, 14:186–201, 2004.

[59] Paul Klemperer. Auction theory: A guide to the literature. *Journal of Economic Surveys*, 13(3):227–86, July 1999.

[60] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[61] Vijay Krishna. *Auction Theory*. Academic Press, 2002.

[62] Kuldeep Kumar. Technology for supporting supply-chain management. *Communications of the ACM*, 44(6):58–61, 2001.

[63] Sebastien Lahaie, David Pennock, Amin Saberi, and Rakesh Vohra. Sponsored search auctions. In Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[64] John Langford and Tong Zhang. The epoch-greedy algorithm for contextual multi-armed bandits. In *Advances in Neural Information Processing Systems 20*, pages 817–824, 2007.

[65] Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *Proceedings of the International Conference in Machine Learning*. Morgan Kaufmann, 2004.

[66] R. D. Lawrence. A machine-learning approach to optimal bid pricing. In *Proceedings of the Eighth INFORMS Computing Society Conference on Optimization and Computation in the Network Era*, Arizona, 2003.

[67] Nick Littlestone and Manfred Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[68] De Liu, Jianqing Chen, and Andrew Whinston. Current issues in keyword auctions. In Gedas Adomavicius and Alok Gupta, editors, *Handbooks in Information Systems: Business Computing*. Emerald, 2009.

[69] Yaxin Liu and Peter Stone. Value-function-based transfer for reinforcement learning using structure mapping. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 415–20, July 2006.

[70] R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, June 1987.

[71] Lilyana Mihalkova, Tuyen Huynh, and Raymond Mooney. Mapping and revising Markov logic networks for transfer learning. In *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, pages 608–614, Vancouver, BC, 2007.

[72] Michael Munsey, Jonathan Veilleux, Sindhura Bikkani, Ankur Teredesai, and Martine De Cock. Born to trade: A genetically evolved keyword bidder for sponsored search. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2010.

[73] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[74] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 99, 2009.

[75] V. Papaioannou and N. Cassaigne. A critical analysis of bid pricing models and support tool. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2098–2193, Piscataway, NJ, 2000.

[76] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, May 2001.

[77] Steve Phelps, Peter Mc Burnley, Simon Parsons, and Elizabeth Sklar. Co-evolutionary auction mechanism design. In *Agent Mediated Electronic Commerce IV*, volume 2531 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2002.

[78] Achim Rettinger, Martin Zinkevich, and Michael Bowling. Boosting expert ensembles for rapid concept recall. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 464–469, July 2006.

[79] Matthew Richardson. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of the 16th International World Wide Web Conference*, pages 521–530, 2007.

[80] A. Rogers, E. David, J. Schiff, S. Kraus, and N. Jennings. Learning environmental parameters for the design of optimal english auctions with discrete bid levels. In *AAMAS 2005 Workshop on Agent Mediated Electronic Commerce VII*, 2005.

[81] Alvin E. Roth and Axel Ockenfels. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. *American Economic Review*, 92(4):1093–1103, 2001.

[82] Michael H. Rothkopf and Ronald M. Harstad. Modeling competitive bidding: A critical essay. *Management Science*, 40(3):364–384, 1994.

[83] Michael Rothschild. A two armed bandit theory of market pricing. *Journal of Economic Theory*, 9:185–202, 1974.

[84] Paat Rusmevichientong and David P. Williamson. An adaptive algorithm for selecting profitable keywords for search-based advertising services. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 260–269, 2006.

[85] Maytal Saar-Tsechansky and Foster Provost. Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178, 2004.

[86] N. Sadeh, D. Hildum, D. Kjenstad, and A. Tseng. MASCOT: an agent-based architecture for dynamic supply chain creation and coordination in the Internet economy. *Journal of Production, Planning and Control*, 12(3):211–223, 2001.

[87] Alberto Sardinha, Michael Benisch, Norman Sadeh, Ramprasad Ravichandran, Vedran Podobnik, and Mihai Stan. The 2007 procurement challenge: A competition to evaluate mixed procurement strategies. *Electronic Commerce Research and Applications*, 8(2):106 – 114, 2009.

[88] Robert E. Schapire, Peter Stone, David McAllester, Michael L. Littman, and János A. Csirik. Modeling auction price uncertainty using boosting-based conditional density estimation. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 546–553, 2002.

[89] Jeffrey Schlimmer and Richard Granger. Beyond incremental processing: Tracking concept drift. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 502–507, 1986.

[90] Stefan Schone. *Auctions In the Electricity Market: Bidding When Production Capacity Is Constrained*. Springer, Berlin, 2009.

[91] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227 – 244, 2000.

[92] Galit Shmueli, Wolfgang Jank, Aleks Aris, Catherine Plaisant, and Ben Shneiderman. Exploring auction databases through interactive visualization. *Decision Support Systems*, 42:1521–1538, December 2006.

[93] Durga Shrestha and Dimitri Solomatine. Experiments with AdaBoost.RT, an improved boosting scheme for regression. *Neural Computing*, 18(7):1678–1710, 2006.

[94] Eric Sodomka, John Collins, and Maria Gini. Efficient statistical methods for evaluating trading agent performance. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, pages 770–775, 2007.

[95] James C. Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins APL Technical Digest*, 19:482–492, 1998.

[96] Mihai Stan, Bogdan Stan, and Adina Magda Florea. A dynamic strategy agent for supply chain management. In *Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 227–232, 2006.

[97] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[98] Peter Stone, Michael L. Littman, Satinder Singh, and Michael Kearns. ATTac-2000: An adaptive autonomous bidding agent. *Journal of Artificial Intelligence Research*, 15:189–206, June 2001.

[99] Peter Stone, Robert E. Schapire, Michael L. Littman, János A. Csirik, and David McAllester. Decision-theoretic bidding based on learned density models in simultaneous, interacting auctions. *Journal of Artificial Intelligence Research*, 19:209–242, 2003.

[100] Malcolm Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 943–950, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[101] Shuang Sun, Viswanath Avasarala, Tracy Mullen, and John Yen. PSUTAC: A trading agent designed from heuristics to knowledge. In *AAMAS 2004 Workshop on Trading Agent Design and Analysis*, 2004.

[102] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[103] Matthew E. Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005.

[104] S. Thrun and L. Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, November 1997.

[105] Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, 25:1163–1178, 2007.

[106] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[107] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[108] Yevgeniy Vorobeychik. A game theoretic bidding agent for the Ad Auction Game. In *Third International Conference on Agents and Artificial Intelligence*, 2011.

[109] Robert J. Weber. Making more from less: Strategic demand reduction in the FCC spectrum auctions. *Journal of Economics and Management Strategy*, 6(3):529–548, 1997.

[110] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[111] David H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[112] Peter Wurman, Michael Wellman, and William Walsh. A parameterization of the auction design space. *Journal of Games of Economic Behavior*, 35:304–338, 2001.

[113] Robert Zeithammer. Forward-looking bidding in online auctions. *Journal of Marketing Research*, 43(3):462–476, 2006.

[114] Yunhong Zhou and Victor Naroditskiy. Algorithm for stochastic multiple-choice knapsack problem and application to keywords bidding. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 1175–1176, New York, NY, USA, 2008. ACM.

Vita

David Pardoe was born in Plano, Texas and attended Plano East Senior High School. He graduated from Brigham Young University in 2002 with a double major in computer science and mathematics, and started his graduate studies at The University of Texas at Austin that year. While a Ph.D. student, he worked as a teaching assistant and graduate research assistant and was a summer intern at Susquehanna International Group.

Permanent address: 4401 Great America Parkway 2GA-3
Santa Clara, CA 95054
dpardoe@cs.utexas.edu
`http://www.cs.utexas.edu/~dpardoe`

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.