In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 871â€“878, Madrid, Spain, July 2015.

Evolutionary Bilevel Optimization for Complex Control Tasks

Jason Liang and Risto Miikkulainen
Dept. of Computer Science, University of Texas at Austin
Austin, Texas, United States
jasonzliang@utexas.edu, risto@cs.utexas.edu

ABSTRACT

Most optimization algorithms must undergo time consuming parameter adaptation in order to optimally solve complex, real-world control tasks. Parameter adaptation is inherently a bilevel optimization problem where the lower level objective function is the performance of the control parameters discovered by an optimization algorithm and the upper level objective function is the performance of the algorithm given its parametrization. In this paper, a novel method called *MetaEvolutionary Algorithm* (MEA) is presented and shown to be capable of efficiently discovering optimal parameters for neuroevolution to solve control problems. In two challenging examples, double pole balancing and helicopter hovering, MEA discovers optimized parameters that result in better performance than hand tuning and other automatic methods. Bilevel optimization in general and MEA in particular, is thus a promising approach for solving difficult control tasks.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Control methods

Keywords

Genetic Algorithms;Metaheuristics;Neural Networks;Fitness Approximation;Parameter Tuning

1. INTRODUCTION

For many traditional control tasks that involve few state variables, classical approaches such as PID controllers are sufficient [1]. However, many modern control problems involve a large number of state of variables that interact in a nonlinear manner, making it difficult to apply classical methods to them [2]. A common way to solve such problems is to pose the control task as a reinforcement learning (RL) problems where the goal is find an optimal policy function that maps states to actions [3]. Given the state of the system as input, an optimal policy outputs a control action that maximizes the performance of the system with respect to some reward criteria.

Unfortunately, traditional RL methods are based on tabular representations and dynamic programming, making it difficult to extend these methods to large, partially observable continuous state spaces in many modern control problems. Recently, much progress has been made in using policy search methods to solve such control tasks [4, 5, 6]. Instead of trying to compute a Q-value for each possible combination of state and action, the policy function is parametrized as a fixed set of parameters p and optimization algorithms are used to find $\text{argmax}_p E(\pi(p))$, where $E(\pi)$ is the expected reward obtained by following the policy π. For many control problems, the fitness landscape described by $E(\pi)$ is non-convex and sometimes even non-differentiable, which means gradient descent is intractable and heuristic based approaches such as evolutionary algorithms (EA) must be used instead [7, 8]. This paper builds on a particular approach called neuroevolution (NE) [9, 10, 11] that combines EAs and neural networks to search for an optimal policy. In NE, the policy is encoded by a neural network where the input is the state vector and the output is an action vector. A population of potential weight vectors for the neural network is then formed and operators such as selection, mutation, and crossover are used to improve the overall fitness of the population over time. EAs have been demonstrated to perform well in many complex control problems, such as finless rocket stabilization, race-car driving, bipedal walking, and helicopter hovering [12, 13, 14, 15].

One major issue with EAs is that their performance is highly sensitive to the parameters used. Even more so than gradient based optimization algorithms, incorrectly set parameters not only make EAs run slowly but make it difficult for them to converge onto the optimal solution. A commonly used and yet vastly inefficient parameter adaptation method is grid search [16], where the parameter space is discretized and all possible combinations of parameters are exhaustively evaluated. Unfortunately, the computational complexity of grid search increases exponentially with the dimensionality of the parameters and is intractable once the number of parameters exceed three or four.

In this paper, the issues with grid search are avoided by framing parameter adaptation as a bilevel optimization problem [17]. Formally, a bilevel problem has two levels: an upper level optimization task with parameters p_u and objective function F_u, and a lower level optimization task with parameters p_l and objective function F_l. The goal is find a p_u that allows p_l to be optimally solved:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

GECCO '15, July 11–15, 2015, Madrid, Spain

© 2015 ACM. ISBN 978-1-4503-3472-3/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739480.2754732

$$\begin{aligned} &\underset{p_u}{\text{maximize}}\ F_u(p_u) = E[F_l(p_l)|p_u] \\ &\text{subject to } p_l = O_l(p_u), \end{aligned} \tag{1}$$

where $E[F_l(p_l)|p_u]$ is the expected performance of the lower level solution p_l obtained by lower level optimization algorithm O_l with p_u as its parameters. The maximization is done by a separate upper level optimization algorithm O_u. In this paper, O_u is the proposed MEA, p_u are the parameters for NE, O_l is NE, p_l are the weights of the neural network, and F_l measures performance in two simulated and separate control tasks: double pole balancing and helicopter hovering. As will be shown, the right heuristics for O_u allow optimal parameters for O_l to be found much faster than with grid search. Specifically, MEA makes use of fitness approximation to reduce the number of F_u evaluations by O_u. Thus the key contributions of this paper are:

1. Casting parameter adaptation for neuroevolution as a blievel optimization problem.
2. Presenting an efficient upper level algorithm O_u that achieves best results to date on two real-world control tasks.
3. Showing that a more complex parametrization of O_l can increase its potential performance.

The rest of the paper is organized as follows: Related work on parameter adaptation is first summarized and then the MEA algorithm is described in detail. MEA is evaluated in the double pole balancing and helicopter hovering tasks, comparing its performance to other hand-designed and automatic parameter adaptation methods.

2. RELATED WORK

Automatic parameter adaptation is a widely studied problem with extensive previous research done in the area. A survey and systematic categorization of current existing techniques is provided by Eiben and Smit [18]. Many conventional methods for parameter adaptation are fundamentally based on the grid search approach (evaluating all possible parameter configurations) with additional heuristics to reduce computational complexity by a constant factor [16, 19]. These include racing algorithms that try to aggressively prune out bad parameters [20]. However, such approaches quickly become intractable as the number of parameters to be tuned increases beyond three or four.

Modern algorithms for parameter adaptation focus on intelligently searching through the space of parameter configurations. One such example is ParamILS [21], which combines hill climbing with restart heuristics. Recently, metaevolutionary algorithms have been developed for solving the parameter adaptation problem [22, 23, 24, 25]. In metaevolution, the upper level algorithm O_u is an EA and the objective function of O_u is the performance of O_l, another EA that solves the target problem. Although more efficient than brute-force search, metaevolution is inherently a nested procedure and still computationally complex. To address this issue, GGA [25] divided the upper level population into two genders and only evaluated a subset of the population every generation. Furthermore, because EA's are stochastic, it is only possible to estimate the performance

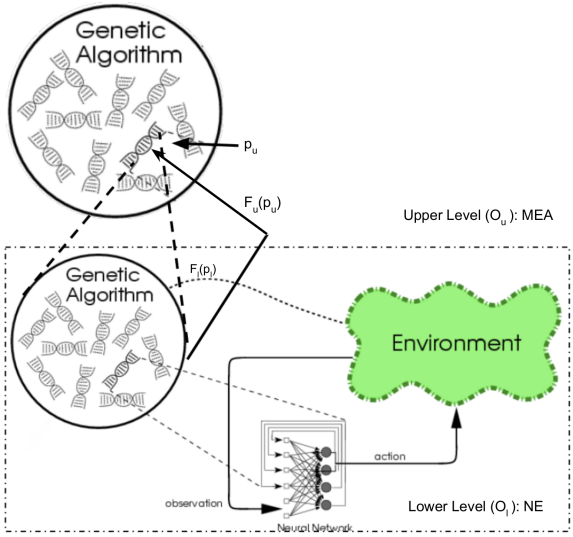

Figure 1: Overview of MEA (O_u) and how it is used for parameter adaptation of NE (O_l). The population in O_u consists of individuals that represent various possible p_u for O_l. To determine $F_u(p_u)$, i.e. the fitness of an upper level individual, $O_l(p_u)$ is run repeatedly q times. During each run, O_l terminates once a fixed number of evaluations $f_{\max}$ is reached. $F_u(p_u)$ is then set as the average fitness of the best solutions returned by all the runs of O_l. The individuals in O_l consists of weights for a neural network and the network is used as the policy for the control task. The network takes as input the state vector of the system being controlled and outputs an action vector that updates the state of the system for the next time step. This process repeats until a termination criterion of the control simulation is met and a fitness value $F_l(p_l)$, measuring performance in the control task, is then assigned to the lower level individual p_l.

of O_l by averaging results from multiple runs. Diosan and Oltean attempted to alleviate this problem by not optimizing p_u but instead optimizing the order in which genetic operators such as mutation and crossover are applied to the population [23].

Metaevolutionary algorithms are closely related to but distinct from self-adaptive EAs [26]. The main difference is that metaevolution attempt to find optimal parameters in an offline environment whereas self-adaptive EAs do so online. Self-adaptive EAs thus can change their parameters dynamically based on the state of the population [26]. On the other hand, they run slower than EAs with static, adapted parameters.

Most current research into metaevolutionary algorithms focuses on fitness function approximation to reduce computational complexity [24, 17]. $F_u(p_u)$ is not always determined by running $O_l(p_u)$, but is sometimes estimated using a regression model. Numerous models have been proposed, including both parametric and nonparametric ones [27, 28, 29, 30]. BAPT, the most recent such algorithm [17], uses a localized quadratic approximation model and achieves good

results on classic test problems. However, BAPT has not been tested on real-world problems and in problems with more than three parameters in p_u. Such more challenging problems are the focus of this paper.

Algorithm 1: MEA

1. Randomly initialize a population of K individuals. Set $F_u(p_u)$ of individuals to actual fitness from evaluating $O_l(p_u)$. Add them to archive Z, and fit regression model R to Z.
2. Create X new individuals via tournament selection, uniform crossover, and uniform Gaussian mutation.
3. Sort population by fitness, and replace X worst individuals in population with the new individuals.
4. Set $F_u(p_u)$ of new individuals to approximate fitness predicted by R.
5. Sort population by fitness again. Set $F_u(p_u)$ of top Y individuals to actual fitness from evaluating $O_l(p_u)$. Add them to Z, and fit R to Z.
6. Repeat from Step 2 until the total number of lower level evaluations by O_l exceeds $t_{\max}$. Return best individual in population.

3. ALGORITHM DESCRIPTION

MetaEvolutionary Algorithm (MEA) is fundamentally a real valued genetic algorithm [31], a type of EA that uses genetic operators such as selection, mutation, crossover, and replacement to improve the overall fitness of a population of solutions, represented by vectors of real numbers, iteratively over generations. NE is another real-valued GA that evolves weights for a neural network with a fixed topology. MEA serves as the upper level algorithm O_u while NE serves as the lower level algorithm O_l. The roles of both MEA and NE within the bilevel optimization framework are described in detail by Fig. 1. A step by step summary of MEA is given by Algorithm 1.

In BAPT [17], $F_u(p_u)$ is sometimes approximated using a regression model in order to reduce the number of times $O_l(p_u)$ is called. However, BAPT's model assumes that the fitness landscape can be described by a quadratic function and requires a large number of data points to avoid overfitting. In contrary, MEA uses a regression model based on randomized decision trees, also known as Random Forests [32]. Random Forests are used for two main reasons: (1) They are nonparametric and make no prior assumptions regarding the shape of $F_u(p_u)$. (2) Random Forests are robust to overfitting [33]. Furthermore, in preliminary experiments, Random Forests gave more accurate approximations, especially when the number of data points is small.

There are two undesirable side effects of fitness approximation: (1) The approximate fitness estimated by the regression model is noisy and (2) as the dimensionality of p_u increases, the archive Z and population K must increase to avoid overfitting. For example in BAPT, which uses R to fit p_u directly, K must increase quadratically with the dimension of p_u [17], which makes it difficult to scale up the approach.

To deal with (1), Step 5 is included in MEA to ensure that the actual fitness of promising but approximately evaluated individuals is always known. In addition, MEA deals with issue (2) by replacing p_u with performance metrics that remain fixed in dimensionality. $O_l(p_u)$ is run for roughly $f_{\max}/\alpha$ ($\alpha > 1$) evaluations and the following six metrics m_u regarding the population of O_l are collected: (1) best fitness, (2) average fitness, (3) fitness standard deviation, (4) population diversity, (5) increase in average fitness over last 10 generations and (6) exact number of function evaluations, which can vary depending on K. Finally, R is fitted using the m_u and actual fitness of individuals in Z, averaged over q runs. The metrics m_u can be used to predict the performance of $O_l(p_u)$ since there is usually strong correlation between an algorithm's initial and final performance. Because the size of m_u is independent of p_u, MEA has no problems dealing with high-dimensional p_u.

As an additional way of smoothing noise, if an upper level individual remains in the population for multiple generations, its fitness is averaged over multiple evaluations. Constraints for p_u are enforced by clipping them to the desired minimum and maximum values after mutation is performed. The archive Z has a maximum size to limit the computational costs of fitting R and when it is full, the newly added individuals replace the oldest ones. MEA terminates automatically when the total number of lower level fitness evaluations by O_l exceeds $t_{\max}$ and returns the p_u with highest fitness. If there is a tie, the individual that remained longest in the population is returned.

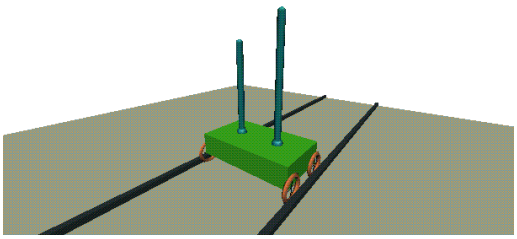

Figure 2: The double pole balancing task. The system consists of two vertical poles with different lengths that are attached to a movable cart with one degree of freedom. The goal is to apply a force to the cart at regular time intervals to ensure that both poles remain upright and that the cart remains within fixed distance of its starting position. The six state variables of the system are the angular position and velocities of the poles, as well as the position and velocity of the cart. The fitness of the controller (F_l) is t_s, the number of time steps it can prevent either pole from falling over or the cart from straying too far away. A pole balancing episode/evaluation is terminated when $t_s = 100,000$ or when failure occurs.

4. EXPERIMENTAL RESULTS

In order to judge the performance of MEA better, it is compared against two other EAs designed for parameter adaptation on two real-world control tasks. The first one is SMEA, a simplified version of MEA that does not perform any fitness function approximation. Instead of performing Steps 4 and 5, the actual fitness of every individual in the population is evaluated before proceeding to the next generation. The second EA is a version of BAPT designed for comparison with MEA and thus named BAPTM; it has the same localized quadratic approximation scheme as BAPT

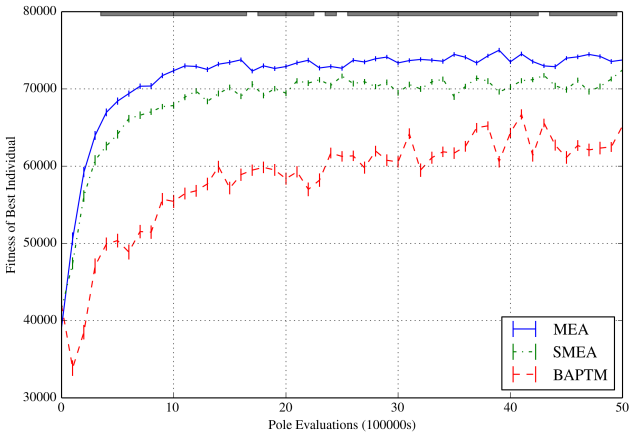

Figure 3: The fitness of the best upper level individual p_u over the number of lower level evaluations F_l. Each plot is an average over 150 O_u runs, where the best individual in the population each generation is retested with another independent set of 30 evaluations. Vertical bars show standard error while black lines at top of plot indicate regions where differences between MEA and SMEA are statistically significant (p-value < 0.05). MEA overtakes SMEA in performance after 500,000 evaluations.

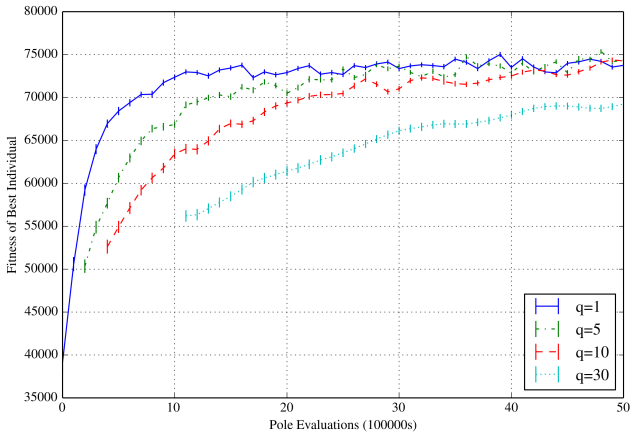

Figure 4: The fitness of the best p_u over the number of evaluations of F_l for different values of q with MEA. The data is gathered in the same manner as Fig. 3. Lower values of q result in better performance, suggesting that MEA is surprisingly robust against evaluation noise.

but uses MEA's uniform crossover in place of PCX crossover [34]. Uniform crossover is a commonly used operator that is shown to be effective in many domains [35, 36], including preliminary experiments in double pole balancing and helicopter hovering. For all experiments except where indicated, the maximum size of Z is set to $100K$, $K = 20$, $X = K/2$, $Y = K/2$ and $\alpha = 4$. Because of an elitist replacement scheme and constraints on p_u, a high mutation rate of 0.5 can be utilized, along with a crossover rate of 0.5. As mentioned in the introduction, NE is utilized as the lower level O_l since it is well suited for finding robust, efficient solutions to difficult control problems [12, 13, 14, 15].

4.1 Double Pole Balancing

Pole balancing, or inverted pendulum, has long been a standard benchmark for control algorithms [37, 38, 3, 7]. It is easy to describe and is a surrogate for other real-world control problems such as finless rocket stabilization. While the original version with a single pole is too easy for modern methods, the double-pole version can be made arbitrarily hard, and thus serves as a good first benchmark for bilevel optimization. A detailed description of the double pole balancing task is given by Fig. 2.

For O_l, a conventional NE algorithm common in the literature [39] is used and the label PNE refers to this NE with hand-designed parameters. The evolved genotype is a concatenation of real-valued weights of a feedforward neural network (with five hidden units) and PNE performs 1-point crossover and uniform mutation. In the first experiment (PNE_5), p_u adapts five of its parameters; in the second (PNE_{15}), 15. The details of these parameters are given in the Appendix.

For PNE_5, the following parameters are set for each O_u: $q = 1$, $f_{\max} = 2,000$, and $t_{\max} = 5,000,000$. $K = 26$ is used for BAPTM as the minimum population size required for optimizing the five parameters of p_u [17]. In Fig. 3, the performances of all three O_u with PNE_5 as O_l are compared. MEA achieves statistically significant higher fitness than SMEA after approximately 500,000 lower level evaluations; both are significantly better than BAPTM in both metrics. Fig. 4 shows how the performance of MEA is affected by different values of q, the number of times $O_l(p_u)$ is evaluated. Interestingly, lower values of q result in faster learning with no difference in peak fitness achieved.

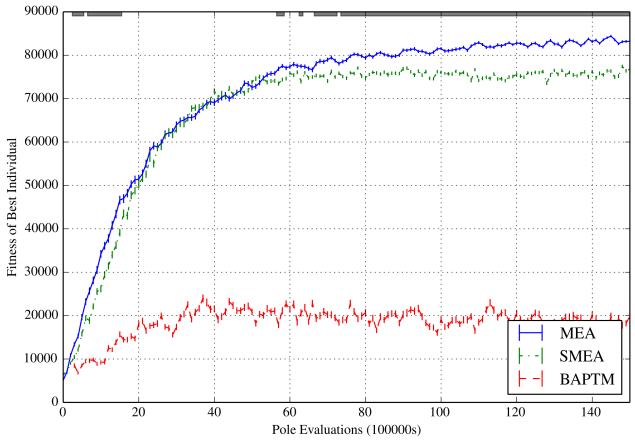

Figure 5: Similar plot as Fig. 3, but with PNE_{15} as O_l. Both MEA and SMEA perform better than BAPTM by a wide margin. MEA begins to overtake SMEA in performance after roughly six million lower level evaluations.

PNE_{15} is a generalization of PNE_5, with 10 of the hand selected parameters of PNE_5 replaced by adaptable parameters. These 10 additional parameters allow the EA to choose what selection and crossover operators to use. The follow-

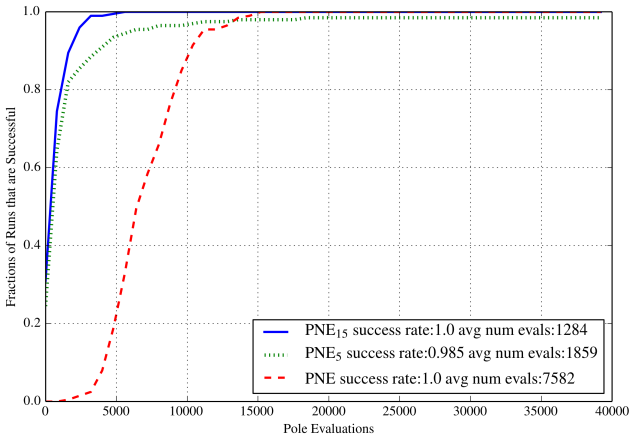

Figure 6: Cumulative histogram of success rate of PNE, PNE_5, and PNE_{15} over the number of evaluations of F_l, with success defined as balancing the poles for t_s time steps. Results are from 200 independent runs of each O_l. All runs solved the task in significantly less time than $f_{\max} = 40,000$ evaluations. PNE_{15} is 1.5 times faster than PNE_5, and five times faster than PNE.

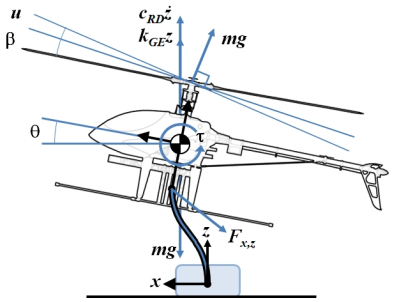

Figure 7: The helicopter hovering task. The goal is to steer a simulated helicopter to remain as close as possible to a fixed point in 3D space. There are 12 state variables that describe the helicopter's position, rotation, linear/angular velocities and four continuous action variables which determine the pitch of the aileron, elevator, rudder, and main rotor. The system takes into account a constant wind velocity (chosen from 10 preset velocities) and sensor noise. The simulation lasts for 60,000 time steps and $F_l(p_l) = 1/\log(X)$ where X is the sum of (1) the cumulative deviation error of the helicopter from its starting position and (2) a additional large penalty if the helicopter exceeds certain bounds on its position and crashes.

ing parameters are set for each O_u : $K = 30$, $q = 1$, $f_{\max} = 2,000$, and $t_{\max} = 15,000,000$. For BAPTM, $K = 151$ as the minimum required size for 15 parameters. The performance of all three O_u are compared in Fig. 5. While MEA and SMEA learn equally fast initially, the difference between the two become statistically significant after six million lower level evaluations and MEA terminates with a higher peak fitness. BAPTM only achieves a fraction of the peak fitness of MEA and SMEA.

So far, the results of PNE_5 and PNE_{15} only show their performance when evaluated up to $f_{\max}$. It remains to be shown that they can consistently and successfully evolve networks that balance poles for $t_s = 100,000$ time steps. Thus, $f_{\max}$ is increased to $40,000$ evaluations and O_l is run 200 times with the parameters p_u of the median performing individual returned by MEA from its 150 runs. As Fig. 6 shows, PNE_{15} performs better than PNE_5 and both PNE_{15} and PNE_5 perform much better than hand-designed PNE, verifying that the optimal p_u are indeed discovered in the full double pole balancing task.

4.2 Helicopter Hovering

The goal in the helicopter hovering task [15] is to keep a simulated helicopter in a fixed position in the air subject to a constant wind velocity. Compared to double pole balancing, helicopter hovering is a significantly more difficult task due to the larger state and action space, sensor noise, complex dynamics, and a fitness function that punishes any deviation from the optimal control policy (Fig. 7).

For O_l, a specialized NE algorithm that has additional mutation replacement and crossover averaging operators [15] is used. The original version with hand-designed parameters is referred to as HNE and the version of this algorithm that adapts eight parameters (specified in the Appendix), is referred to as HNE_8. The network evolved by HNE has a hand-designed topology and consists of 11 linear and sigmoid hidden units [15]. The following parameters are set for each O_u: $q = 1$, $f_{\max} = 5,000$, and $t_{\max} = 15,000,000$; for BAPTM, $K = 51$. As seen in Fig. 8, the performance improvement of MEA over SMEA becomes statistically significant after roughly four million evaluations and both are significantly better than BAPTM. Fig. 9 compares the performance of representative examples (median performing individual for each wind velocity) of HNE_8 and HNE when the task is solved fully with $f_{\max} = 25,000$. In comparison to HNE, HNE_8 learns faster and achieves higher peak fitness, demonstrating the value of bilevel optimization over hand-design.

5. DISCUSSION AND FUTURE WORK

The results from the experiments show that MEA performs better than SMEA and that both are much better than BAPTM. The main cause of BAPTM's poor performance are the inaccurate approximations given by its regression model. The relatively few number of individuals in Z during the first generations and noise in evaluating $O_l(p_u)$ cause the quadratic model to overfit initially. Furthermore, the fitness landscape of F_u is irregular and not well described by a quadratic approximation. For example, preliminary experiments show that doubling the population size of PNE_5 can result in $F_u(p_u)$ decreasing by an order of magnitude.

Surprisingly, running MEA with lower values of q results in better performance and faster convergence to the optimal fitness value. This result is counter-intuitive because there is more evaluation noise with smaller values of q and some averaging of evaluations should be useful. One explanation is that as the number of generations increase, both the fitness averaging and approximation mechanisms of MEA become more accurate. They eventually help smooth out the evaluation noise, retaining individuals with good fitness, and elim-

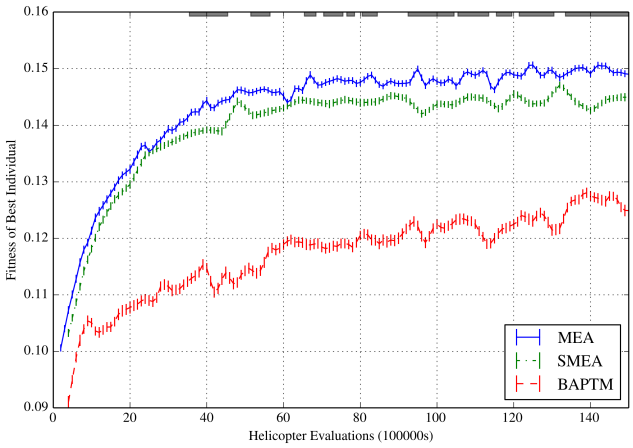

Figure 8: Similar plot as Fig. 3, but with HNE_8 as O_l. O_u is run with each of the 10 wind velocities 15 times, for a total of 150 runs. Because the helicopter simulation is more computationally complex, data is only collected after every fourth generation. Both MEA and SMEA perform better than BAPTM by a wide margin. MEA begins to overtake SMEA in performance after roughly four million lower level evaluations.

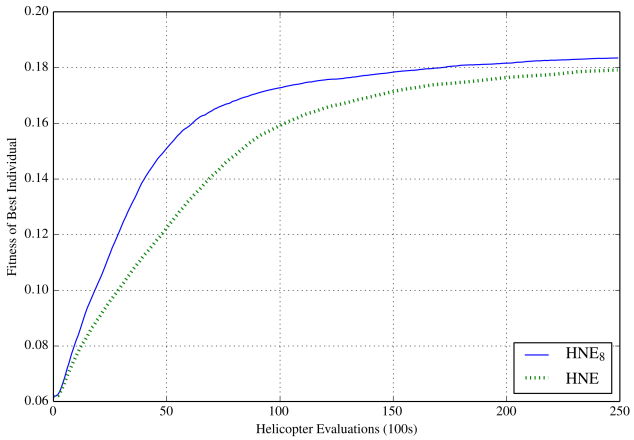

Figure 9: The fitness of the best individual over the number of evaluations of F_l by HNE and HNE_8. Results are averaged over 500 runs, with 50 runs for each of the 10 wind velocities. HNE_8 both learns faster and achieves higher peak fitness than HNE.

inating those with bad fitness. This conclusion is supported by observations that the performance of MEA steadily becomes less noisy and more consistent over the generations. As seen in Figs. 3, 5, and 8, MEA's fitness approximation mechanism also explains why its performance continues to increase even SMEA, which does not use fitness approximation, is flattening out.

Interestingly, although PNE_{15} has three times as many p_u as PNE_5, the performance of PNE_{15} is noticeably better. A more complex parametrization of an optimization algorithm allows it to adapt better to a problem, especially if it is using parameters discovered through bilevel optimization. With MEA, it might thus be possible to discover specialized optimization algorithms that excel in solving particular problem domains.

There are several interesting directions of future work: (1) Evolving a policy for an EA that, based on the current state of the population, adaptively changes parameters such as population size, mutation, and crossover rate. Preliminary results on multimodal objective functions such as Shekel's foxholes show that an effective policy (parametrized as weights of a neural network) can be discovered through metaevolution. (2) Optimizing not only the parameters of NE at the upper level, but also the topology of the neural network. (3) Combining various heuristics from existing EAs, such as differential evolution and particle swarm optimization, to create a general purpose lower level parametrization that performs well on a wide variety of optimization problems.

6. CONCLUSION

In this paper, parameter adaptation for NE is cast as a bilevel optimization problem. A novel upper level optimization algorithm, MEA, is proposed and shown to be capable of achieving better results in optimizing parameters for NE. The optimized parameters discovered by MEA result in significantly better performance over hand-designed ones in two difficult real-world control tasks. Remarkably, evolving a more complex parametrization of the lower level optimization algorithm results in better performance, even though such a parametrization would be very difficult to manage by hand. Bilevel optimization is thus a promising approach for solving complex control tasks.

7. APPENDIX

For the double pole balancing task, the length and mass of the longer pole is set to 0.5 meters and 0.1 kilogram. The length and mass of the shorter pole is set to 0.05 meters and 0.01 kilogram. The mass of the cart itself is 1.0 kilogram. The shorter pole is initialized in a vertical position while the longer pole is initialized with one degree offset from the shorter pole. Both poles and the cart have zero initial velocity. For the helicopter hovering task, the physical dynamics of the helicopter simulator is described in more detail in [15].

The table below lists the parameter names, constraints, and default values for the p_u of PNE_5, PNE_{15}, and HNE_8. At the beginning of every generation in PNE_{15}, the probability a particular selection or crossover operator is chosen to help generate next generation's population is proportional to the probability parameter assigned to that operator. In HNE_8, mutation replacement rate is the probability that a randomly generated value replaces a particular gene instead of being added to it. Crossover averaging rate is the probability that corresponding genes in two individuals are averaged together instead of being swapped. The meaning of the other parameters should be clear from their names.

PNE_5 Parameters	**Constraints**	**Default**
Mutation Rate	[0,1]	0.40
Mutation Amount	[0,1]	0.30
Replacement Fraction	[0,1]	0.50
Initial Weight Range	[0,12]	6.00
Population Size	[0,400]	400
Extra PNE_{15} Parameters	**Constraints**	**Default**
Mutation Probability	[0,1]	1.00
Crossover Probability	[0,1]	1.00
Uniform Crossover Rate	[0,1]	0.00
Tournament/Truncation Fraction	[0,1]	0.25
Tournament Selection Probability	[0,1]	0.00
Truncation Selection Probability	[0,1]	1.00
Roulette Selection Probability	[0,1]	0.00
1-pt Crossover Probability	[0,1]	1.00
2-pt Crossover Probability	[0,1]	0.00
Uniform Crossover Probability	[0,1]	0.00
HNE_8 Parameters	**Constraints**	**Default**
Mutation Probability	[0,1]	0.75
Mutation Rate	[0,1]	0.10
Mutation Amount	[0,1]	0.80
Mutation Replacement Rate	[0,1]	0.25
Replacement Fraction	[0,1]	0.02
Population Size	[0,200]	50
Crossover Probability	[0,1]	0.50
Crossover Averaging Rate	[0,1]	0.50

To illustrate how evolution improves upon the default values, the table below lists the final parameters evolved by the median performing run of MEA.

PNE_5 Parameters	**Values Found by MEA**
Mutation Rate	0.46
Mutation Amount	0.59
Replacement Fraction	0.87
Initial Weight Range	6.53
Population Size	17
PNE_{15} Parameters	**Values Found by MEA**
Mutation Rate	0.65
Mutation Amount	1.00
Replacement Fraction	1.00
Initial Weight Range	2.67
Population Size	32
Mutation Probability	1.00
Crossover Probability	0.85
Uniform Crossover Rate	0.18
Tournament/Truncation Fraction	0.57
Tournament Selection Probability	0.96
Truncation Selection Probability	0.01
Roulette Selection Probability	0.03
1-pt Crossover Probability	0.00
2-pt Crossover Probability	0.42
Uniform Crossover Probability	0.58
HNE_8 Parameters	**Values Found by MEA**
Mutation Probability	1.00
Mutation Rate	0.03
Mutation Amount	0.77
Mutation Replacement Rate	0.21
Replacement Fraction	0.04
Population Size	28
Crossover Probability	0.94
Crossover Averaging Rate	0.03

8. ACKNOWLEDGMENTS

This research was supported in part by NSF grants DBI-0939454, IIS-0915038, and SBE-0914796, and by NIH grant R01-GM105042.

9. REFERENCES

[1] Karl Johan Åström and Tore Hägglund. *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society; Research Triangle Park, NC 27709, 2006.

[2] Faustino John Gomez and Risto Miikkulainen. *Robust non-linear control through neuroevolution*. Computer Science Department, University of Texas at Austin, 2003.

[3] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[4] Leemon Baird and Andrew W Moore. Gradient descent for general reinforcement learning. *Advances in neural information processing systems*, pages 968–974, 1999.

[5] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1615–1620. IEEE, 2001.

[6] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2619–2624. IEEE, 2004.

[7] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *The Journal of Machine Learning Research*, 9:937–965, 2008.

[8] John J Grefenstette, David E Moriarty, and Alan C Schultz. Evolutionary algorithms for reinforcement learning. *arXiv preprint arXiv:1106.0221*, 2011.

[9] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[10] Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[11] Joel Lehman and Risto Miikkulainen. Neuroevolution. *Scholarpedia*, 8(6):30977, 2013.

[12] Faustino J Gomez and Risto Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Genetic and Evolutionary Computation-GECCO 2003*, pages 2084–2095. Springer, 2003.

[13] Julian Togelius and Simon M Lucas. Evolving controllers for simulated car racing. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1906–1913. IEEE, 2005.

[14] Brian F Allen and Petros Faloutsos. Evolved controllers for simulated locomotion. In *Motion in Games*, pages 219–230. Springer, 2009.

[15] Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary intelligence*, 4(4):219–241, 2011.

[16] Richard Myers and Edwin R Hancock. Empirical modelling of genetic algorithms. *Evolutionary computation*, 9(4):461–493, 2001.

[17] Ankur Sinha, Pekka Malo, Peng Xu, and Kalyanmoy Deb. A bilevel optimization approach to automated parameter tuning. 2014.

[18] Agoston E Eiben and Selmar K Smit. Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1):19–31, 2011.

[19] Belarmino Adenso-Diaz and Manuel Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54(1):99–114, 2006.

[20] Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle. F-race and iterated f-race: An overview. In *Experimental methods for the analysis of optimization algorithms*, pages 311–336. Springer, 2010.

[21] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.

[22] John J Grefenstette. Optimization of control parameters for genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 16(1):122–128, 1986.

[23] Laura Silvia Diosan and Mihai Oltean. Evolving evolutionary algorithms using evolutionary algorithms. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2442–2449. ACM, 2007.

[24] Volker Nannen and Agoston E Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *IJCAI*, volume 7, pages 975–980, 2007.

[25] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Principles and Practice of Constraint Programming-CP 2009*, pages 142–157. Springer, 2009.

[26] Oliver Kramer. Evolutionary self-adaptation: a survey of operators and strategy parameters. *Evolutionary Intelligence*, 3(2):51–65, 2010.

[27] Alain Ratle. Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In *Parallel Problem Solving from Nature-PPSN V*, pages 87–96. Springer, 1998.

[28] Iloneide CO Ramos, Marco César Goldbarg, Elizabeth G Goldbarg, and Adrião Duarte Dória Neto. Logistic regression for parameter tuning on an evolutionary algorithm. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1061–1068. IEEE, 2005.

[29] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, 9(1):3–12, 2005.

[30] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[31] Lawrence Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

[32] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[33] Marko Robnik-Šikonja. Improving random forests. In *Machine Learning: ECML 2004*, pages 359–370. Springer, 2004.

[34] Ankur Sinha, Aravind Srinivasan, and Kalyanmoy Deb. A population-based, parent centric procedure for constrained real-parameter optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 239–245. IEEE, 2006.

[35] Gilbert Syswerda. Uniform crossover in genetic algorithms. 1989.

[36] William M Spears and Kenneth D De Jong. On the virtues of parameterized uniform crossover. Technical report, DTIC Document, 1995.

[37] Alexis P Wieland. Evolving neural network controllers for unstable systems. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 667–673. IEEE, 1991.

[38] Hamid R Berenji and Pratap Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *Neural Networks, IEEE Transactions on*, 3(5):724–740, 1992.

[39] Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *Machine Learning: ECML 2006*, pages 654–662. Springer, 2006.