Neuroevolutionary Reinforcement Learning for Generalized Helicopter Control

Rogier Koppejan
Informatics Institute
University of Amsterdam
Science Park 107, 1098 XG
Amsterdam, The Netherlands
r.j.koppejan@uva.nl

Shimon Whiteson
Informatics Institute
University of Amsterdam
Science Park 107, 1098 XG
Amsterdam, The Netherlands
s.a.whiteson@uva.nl

ABSTRACT

Helicopter hovering is an important challenge problem in the field of reinforcement learning. This paper considers several neuroevolutionary approaches to discovering robust controllers for a generalized version of the problem used in the 2008 Reinforcement Learning Competition, in which wind in the helicopter's environment varies from run to run. We present the simple model-free strategy that won first place in the competition and also describe several more complex model-based approaches. Our empirical results demonstrate that neuroevolution is effective at optimizing the weights of multi-layer perceptrons, that linear regression is faster and more effective than evolution for learning models, and that model-based approaches can outperform the simple model-free strategy, especially if prior knowledge is used to aid model learning.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Experimentation

Keywords

neural networks, evolutionary computation, reinforcement learning, robot control

1. INTRODUCTION

The field of *reinforcement learning* (RL) [10, 21] aims to develop on-line algorithms for optimizing behavioral policies in *sequential decision problems* (SDPs), wherein agents interact with unknown environments and seek behavioral policies that maximize their long-term reward. Many challenging and realistic tasks can be cast in this framework (e.g., robot control [18], game-playing [22], and system optimization [24]), so developing effective RL algorithms is critical to the progress of artificial intelligence.

In recent years, in an effort to encourage the community to convert its research ideas into operational methods, RL researchers have begun organizing competitive events. The most recent such event, the 2008 RL Competition, began in November 2007 and ended at the International Conference on Machine Learning (ICML) in July 2008 in Helsinki, Finland. It featured six problems of various size and difficulty, some of which were formulated as *generalized domains*. A generalized domain is not a single SDP but a class of related SDPs that vary along specific dimensions (e.g., sensor noise or environment size). A small set of SDPs drawn from this class were available to participants to use for training but the competition was decided solely on performance on a different set of SDPs independently sampled from the same class. In this way, the competition discouraged participants from overfitting to a single SDP and encouraged them to develop methods capable of robust, on-line learning.

One such generalized domain from the 2008 competition is the problem of *helicopter hovering* [3, 14], in which a 3-dimensional simulated helicopter strives to hover as close as possible to a fixed position. Helicopter hovering is an important RL challenge problem because it has complex transition dynamics, high-dimensional observation and action spaces, and high risk. While the helicopter hovering problem has already been well studied, the 2008 RL Competition marked the first attempts to tackle the generalized version, wherein wind in the helicopter's environment varies greatly from one SDP to the next.

This paper describes and analyzes several neuroevolutionary strategies for tackling the generalized helicopter hovering problem. We present the simple model-free strategy that won first place in the 2008 competition by evolving specialized neural network controllers for each SDP available in the training set. For each testing SDP, the model-free approach automatically determines which specialized controller is most appropriate. We also describe more complex model-based approaches in which the agent uses flight data gathered in early episodes to estimate a model of the environment and then evolves a controller off-line.

Our empirical analysis demonstrates that evolution is effective at optimizing the weights of controllers represented as fixed-topology multi-layer perceptrons. However, neuroevolution is not able to evolve topologies that outperform

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

the best ones manually constructed by human experts. Our results also show that linear regression is faster and more effective than evolution for estimating models from flight data. Finally, our results demonstrate that model-based approaches can outperform the simple model-free strategy, though the size of the performance difference depends on how much prior knowledge about the environment is employed during model learning.

The rest of this paper is organized as follows. Section 2 formalizes the notion of generalized domains and Section 3 describes the generalized helicopter hovering problem. Sections 4 and 5 compare different approaches to evolving neural network controllers and learning models, respectively, for a given helicopter hovering SDP. Section 6 compares model-free and model-based approaches to generalized helicopter hovering. Section 7 discusses these results and Section 8 outlines opportunities for future work.

2. GENERALIZED DOMAINS

To discourage participants from overfitting to a single SDP and to encourage them to develop methods capable of robust, on-line learning, several events in the 2008 RL Competition used generalized domains. In a generalized domain, the SDPs available for training and testing can differ from each other in particular ways. Defining a generalized domain involves specifying the dimensions along which these SDPs can vary and selecting a distribution over the resulting space. The training and testing SDPs are then formed by drawing independent samples from this distribution. Formally, a generalized domain $\mathcal{G} = \langle \Theta, \mathcal{F}, \mathcal{P} \rangle$ consists of:

- Θ, a set of possible parameter vectors. Each parameter specifies something about the dynamics or reward structure of the SDP.
- $\mathcal{F}$, a function mapping parameter vectors to SDPs, i.e., a particular SDP is defined by $\mathcal{F}(\theta)$, where $\theta \in \Theta$, and
- $\mathcal{P} : \Theta \rightarrow [0, 1]$, a probability distribution over the set of possible parameter vectors.

Given a generalized domain $\mathcal{G}$, a training set is formed by sampling k times from $\mathcal{P}$ and applying $\mathcal{F}$ to each sampled parameter vector, yielding the set $\{\mathcal{F}(\theta_1), \ldots, \mathcal{F}(\theta_k)\}$. A testing set can be generated in the same way. At the time of the competition, neither $\mathcal{F}$, Θ, nor $\mathcal{P}$ were known to the participants. Instead, they observed $\mathcal{F}(\theta_i)$ indirectly from the trajectories of observations and rewards that it generated.

To excel in a generalized domain, a learner must be robust to the variation represented by Θ and $\mathcal{P}$. Except in degenerate cases, no fixed control policy will perform well across many settings in Θ. Consequently, for strong performance, real learning is required during testing. No matter how much learning is done on the training set, the agent must still learn on each test run which SDP it faces and therefore what policy will perform well.

3. GENERALIZED HELICOPTER HOVERING

In the helicopter hovering domain, an RL agent seeks a control policy (i.e., a mapping from observations to actions) for a 3-dimensional simulated XCell Tempest helicopter. The goal is to make the helicopter hover as close as possible to a fixed position for the duration of an episode. Helicopter hovering is challenging for several reasons. First, the transition dynamics (i.e., how the true state of the agent's environment changes in response to its actions) are complex. Second, both the observation and action spaces are continuous and high-dimensional. The latter is particularly problematic as many traditional RL algorithms, e.g., *temporal difference methods* [19], require enumerating the action space in order to select a maximizing action. Third, the domain involves high risk, as bad policies can crash the helicopter, incurring catastrophic negative reward. As a result, the process of exploration, in which new policies are evaluated, must be conducted with extreme care. Below we give a brief description of the helicopter hovering problem; more details can be found in [3, 14].

x	x-axis position
y	y-axis position
z	z-axis position
u	x-axis velocity
v	y-axis velocity
w	z-axis velocity
ϕ	rotation around x-axis (roll)
θ	rotation around y-axis (pitch)
ω	rotation around z-axis (yaw)
p	angular velocity around x-axis
q	angular velocity around y-axis
r	angular velocity around z-axis

Table 1: The 12-dimensional state and observation space in the helicopter hovering domain.

a_1	longitudinal cyclic pitch (aileron)
a_2	latitudinal cyclic pitch (elevator)
a_3	tail rotor collective pitch (rudder)
a_4	main rotor collective pitch

Table 2: The 4-dimensional action space in the helicopter hovering domain.

In each timestep, the agent receives a 12-dimensional observation vector (see Table 1) consisting of a noisy description of its true state, which is also 12-dimensional. It responds by specifying a 4-dimensional action (see Table 2).

At each timestep, the agent also receives an immediate reward that is the sum, over all state features, of the squared difference between that state feature and the fixed target position in which the helicopter wishes to hover:

$$\mathcal{R} = -\sum_i (s_i - t_i)^2$$

where s_i and t_i are the current value and the target value, respectively, of the ith state feature.

A helicopter episode consists of 6000 timesteps, each lasting 0.1 seconds, resulting in episodes of 10 minutes. The episode ends prematurely if the helicopter crashes, which occurs if the velocity along any of the main axes exceeds 5m/s, the position is off by more than 20m, the angular rate around any of the main axes exceeds $2 * 2\pi$ rad/s or the orientation is more than 30 degrees from the target orientation. Crashing results in a large reward penalty equal to the most negative reward achievable for the remaining time. The goal of the agent is to find a control policy that maximizes the average reward it receives per episode.

The original helicopter hovering problem has been well studied [3, 14]. However, the 2008 RL Competition featured a new, generalized version of the domain in which wind was added to the helicopter simulator. After the competition, the software was made public[1], revealing exactly how wind affects the helicopter dynamics. The set of possible parameter vectors is:

$$\Theta = \{\langle wind_u, wind_v \rangle : -5m/s \leq wind_u, wind_v \leq 5m/s\}$$

where $wind_u$ and $wind_v$ control the velocity of wind in the x- and y-axes, respectively. The probability distribution $\mathcal{P}$ over this set is uniform. The presence of wind changes the way the helicopter responds to the agent's actions, thus altering the control policy needed to hover. Therefore, to excel in the generalized version of the problem, an agent must reason about the level of wind in each SDP it faces and adapt its behavior accordingly.

4. EVOLVING HELICOPTER POLICIES

Before addressing the challenges of the generalized version of helicopter hovering, we first consider how to learn policies for a single helicopter SDP with fixed wind parameters. This process is a central component in each of the methods presented in Section 6 for tackling the full, generalized problem. We consider a *neuroevolutionary* approach [25], which uses evolutionary computation [6] to optimize a population of neural networks. Given its numerous successes in noisy, continuous control problems [8, 9, 13, 17, 23, 24], neuroevolution is well-suited to this task.

In the helicopter problem, each neural network generated by evolution represents a different policy mapping observations to actions. To find a good policy, we employ a simple *steady-state* neuroevolutionary method that does not use generations; instead, only one member of the population is changed at a time. Table 3 lists the parameters of this algorithm, along with the settings used in all our experiments. These settings were chosen after an informal parameter search. However, we found that performance was not highly sensitive to these parameters and was similar at other reasonable settings.

`pop_size`	50	`mutate_weight_prob`	0.1
`crossover_prob`	0.5	`replace_gene_prob`	0.25
`averaging_prob`	0.5	`mutate_std`	0.8
`mutate_prob`	0.75	`plateau_threshold`	1000

Table 3: Neuroevolution parameter settings.

Each neural network is encoded as a genome consisting of a vector of the network's weights and an initial population of size `pop_size` is formed by repeatedly applying weight mutations to a given prototype network. The probability of a given weight being mutated is given by `mutate_weight_prob`. When weight mutations occur, their magnitude is sampled from a Gaussian distribution with mean 0.0 and standard deviation `mutate_stand_dev`. With probability `replace_gene_prob`, the weight is replaced by this value instead of added to it.

Each network in the initial population is then evaluated using the fitness function. In the helicopter hovering domain, the fitness function consists of the reward that the agent accrues during a single episode when using the policy specified by the network. We also tried using longer fitness evaluations in which reward is averaged over multiple episodes but found no performance improvement.

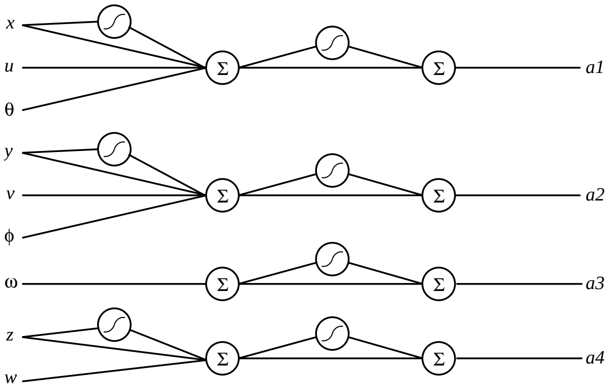

Figure 1: The manually designed topology of a neural network helicopter controller, indicating which nodes use sigmoid activation functions and which use linear summations.

After evaluating the entire initial population, the worst-performing network is repeatedly replaced with a new individual. With probability `crossover_prob`, the new individual is formed via crossover between two parents selected via *roulette wheel selection*. During crossover, each offspring weight is set to the average of its parents' weights with probability `averaging_prob`. Otherwise, it is set equal to the weight of one of the two parents, selected randomly. With probability `mutate_prob`, weight mutations are then applied in the same manner as in the initial population. If crossover does not occur, the new individual is created by applying such weight mutations to a single parent, also chosen with roulette wheel selection. Evolution continues until no new population champion has been discovered in `plateau_threshold` evaluations.

We consider four approaches to evolving neural networks using this method. In the first approach, we evolve fully-connected *single-layer perceptrons* (SLPs), i.e., neural networks without any hidden nodes. In the initial prototype network, all weights are set to 0.0.

In the second approach, we evolve SLPs but starting from a prototype network whose weights correspond to a *baseline controller* provided with the competition software. This baseline controller is robust in that it never causes the helicopter to crash. However, its performance is quite weak, as it is unable to consistently hover near the target point.

In the third approach, we evolve *multi-layer perceptrons* (MLPs) using a topology manually constructed by human experts [14], The topology, shown in Figure 1, employs both sigmoid and linear activation functions.

In the fourth approach, we evolve MLPs but using initial weights that make it equivalent to the baseline controller. To implement the linear baseline controller in this nonlinear topology, all links from the hidden nodes to the outputs are set either to 1.0, to propagate the input signal without modification, or to 0.0, so that they have no influence on the network's output.

All networks have nine inputs, corresponding to the first nine observation features described in Table 1, and four out-

[1] Available at `http://rl-competition.googlecode.com`.

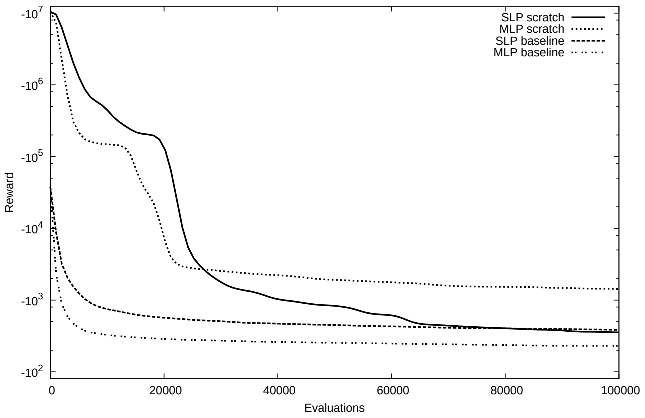

Figure 2: Average performance of the population champion over time (lower is better).

puts, one for each action feature. The angular rates (p, q and r) are omitted because they can be derived from the other nine observation features.

In all of these approaches, evolution optimizes only the weights of fixed-topology networks. However, we also tested two neuroevolutionary methods, NeuroEvolution of Augmenting Topologies (NEAT) [17] and Alternating Extension and Optimization of Neural Networks (AEONN) [11], that can simultaneously optimize network topologies and weights. Unfortunately, neither of these methods discovered topologies that outperformed the manually designed topology. We suspect this result is due to both the quality of the topology, which was highly engineered, and the size of the topology space, which makes it difficult for evolution to search.

To compare these four approaches, we conducted 10 independent runs of each approach in each of the 10 training SDPs, i.e., 100 runs per approach and 400 runs total. The results, averaged over all 100 runs for each method, are shown in Figure 2.

The results demonstrate that the baseline controller, by enabling evolution to begin in a relatively fit region of the search space, can significantly speed evolution. The results also show that, when the baseline controller is used, the manually designed MLP performs substantially better than the SLP. This is not surprising since the topology was carefully engineered for helicopter control. More surprising is the poor performance of the MLP when beginning from scratch, without the aid of the baseline controller. This strategy appears to perform the worst. However, a closer examination of the individual runs revealed that the vast majority achieve performance similar to the MLP using the baseline controller, albeit more slowly. The remaining few runs converge prematurely, performing badly enough to greatly skew the average.

All the approaches described in this section evolve policies only for a single training SDP, with no attempt to generalize across SDPs with different wind settings. To determine the robustness of the resulting policies, we compared their average performance across all 10 training SDPs to their performance on the particular SDP for which they were trained. Specifically, we selected the best single-layer and multi-layer policy evolved for each SDP and tested it for 10 episodes in every training SDP. The results are shown in Table 4.

This comparison demonstrates that the MLP policies are far more robust, achieving much better average performance and lower variance across the training SDPs. In fact, no specialized multi-layer policy crashes the helicopter on any of the 10 SDPs. By contrast, the single-layer policies frequently crash on SDPs other than those they trained on, with catastrophic effects on average reward. Nonetheless, even the MLPs see an order of magnitude performance drop when tested across all training SDPs. This result underscores the challenges of achieving high performance in the generalized version of the task, for which we consider several approaches in Section 6.

Topology	r	σ	$r_{\mathcal{G}}$	$\sigma_{\mathcal{G}}$
SLP	-496.22	25.00	-2.508e6	2.345e5
MLP	-132.60	2.17	-2001.89	46.43

Table 4: Performance of the best SLP and MLP policies: average reward (r) and standard deviation (σ) on the particular SDPs for which they were trained and average reward ($r_{\mathcal{G}}$) and standard deviation ($\sigma_{\mathcal{G}}$) across all training SDPs.

5. LEARNING HELICOPTER MODELS

The results presented above demonstrate that neuroevolution can discover effective policies for helicopter hovering. However, doing so has high *sample complexity*, i.e., it requires evaluating tens of thousands of policies through interactions with the environment. Many of these policies yield poor reward or even crash the helicopter. Consequently, evolving policies on-line is infeasible for the competition because participants are evaluated on the cumulative reward their agents accrue during learning.

One way to reduce sample complexity is to learn a model of the environment. Given an observation at timestep t, a model predicts the observation and reward at timestep $t + 1$. If the agent can learn a model from flight data for each testing SDP early in the run, that model can simulate the fitness function required by neuroevolution. Thus, once a model has been learned, evolution can proceed off-line without increasing sample complexity. Doing so allows the agent to employ a good policy much earlier in the run, thus increasing its cumulative reward.

This section presents several strategies for learning models in the generalized helicopter hovering domain. Section 5.1 describes approaches for learning a complete model based on a linear representation from the helicopter literature, using either evolution or linear regression to optimize the weights. These approaches rely only on information that was publicly available to participants during the competition. Section 5.2 considers how to learn models given additional information gleaned from the post-competition release of the simulator software, which revealed how different wind settings affect the helicopter dynamics. Given this information, we consider strategies for estimating wind settings in a given SDP using either evolution or linear regression.

5.1 Learning Complete Models

During the competition, the details of the helicopter environment were hidden. However, helicopter dynamics have been well studied. In particular, Abbeel et al. [1] developed a representation of the transition function of a hovering helicopter that uses a set of linear equations to predict accelerations given an observation and action at time t.

$$\begin{aligned}
u_{t+1} - u_t &= C_u u_t + g_u + D_u + w_u \\
v_{t+1} - v_t &= C_v v_t + g_v + D_v + w_v \\
w_{t+1} - w_t &= C_w w_t + g_w + C_{a_4} a_{4t} + D_w + w_w
\end{aligned}$$

$$\begin{aligned}
p_{t+1} - p_t &= C_p p_t + C_{a_1} a_{1t} + D_p + w_p \\
q_{t+1} - q_t &= C_q q_t + C_{a_2} a_{2t} + D_q + w_q \\
r_{t+1} - r_t &= C_r r_t + C_{a_3} a_{3t} + D_r + w_r
\end{aligned}$$

The accelerations depend on the values of the $\vec{g}$ vector, which represent gravity ($9.81 m/s$) expressed in the helicopter frame. The values of $\vec{w}$ are zero-mean Gaussian random variables that represent perturbations in acceleration due to noise.

Integrating the accelerations produces an estimate of the velocities at time $t+1$. These velocities describe half the observation at time $t+1$. The remaining half, which describes the helicopter's position, is estimated by adding the velocities at time t to the position at time t. As described in Section 3, reward is a simple function of the helicopter's current state, which can be approximated using the observation estimated with these equations.

Because this model representation was not designed for the generalized version of the problem, it does not explicitly consider the presence of wind. Nonetheless, it can still produce accurate models if the amount of wind in the helicopter frame remains approximately constant, i.e., when the helicopter position and orientation remain fixed. Since helicopters in the hovering problem aim to keep the helicopter as close to the target position as possible, this assumption holds in practice. Therefore, wind can be treated as a constant and learning a complete model requires only estimating values for the weights $\vec{C}$, $\vec{D}$, and $\vec{w}$.

We consider three different approaches to learning these weights. In the first approach, we use evolutionary computation to search for weights that minimize the error in the reward that the model predicts a given policy will accrue during one episode. This approach directly optimizes the model for its true purpose: to serve as an accurate fitness function when evolving helicopter policies. To do so, we apply the same steady-state evolutionary method described in Section 4. Fitness is based on the error in total estimated reward per episode using a single policy trained on an SDP with no wind, which we call the *generic policy.*

In the second approach, we use evolutionary computation to search for weights that minimize error in the model's one-step predictions. In other words, fitness is based on the average accuracy, across all timesteps t in an episode, of the observation predicted at time $t+1$. Again we use the same steady-state evolutionary method and compute fitness using the generic policy.

In the third approach, we still try to minimize error in one-step predictions but use linear regression in place of evolutionary computation. Linear regression computes the weight settings that minimize the least squared error given one episode of data gathered with the generic policy.

For both the second and third approaches, the flight data must first be preprocessed by dividing it into pairs of consecutive observations and subtracting gravity $\vec{g}$ from the observation at $t+1$. After preprocessing, evolution or linear regression is used to estimate $\vec{C}$ and $\vec{D}$. The noise parameters $\vec{w}$ are approximated using the average of the squared prediction errors of the learned model on the flight data.

Complete Model Learning

Method	t	r_m	r_a	σ
EC-MER	562.94	-1.55e4	-1.184e6	3.268e6
EC-MENO	611.10	-223.19	-4988.82	6722.97
LR	2.05	-142.25	-974.24	305.68

Wind Model Learning

Method	ϵ	t	r_m	r_a	σ
EC-MER	0.009	6506.26	-126.26	-136.53	38.72
EC-MENO	0.018	25.88	-129.66	-142.98	40.80
LR	0.009	1.65	-131.42	-141.01	35.35

Table 5: Performance of complete and wind models learned via evolutionary computation to minimize error in reward (EC-MER) or in the next observation (EC-MENO), or via linear regression (LR). Results compare error in the learned weights (ϵ), average computation time (t) in seconds to learn the model and the median (r_m), average (r_a) and standard deviation (σ) of the reward of the best policy evolved with the model and tested in the SDP for which the model was trained.

We evaluated each of these approaches by using them to learn models for each of the 100 test SDPs that were released after the competition ended. Then we used the learned model to evolve policies in the manner described in Section 4. Finally, we took the best policy discovered in each evolutionary run and averaged its performance over 5 episodes in the SDP on which the corresponding model was trained.

Results are shown in the top part of Table 5. Note that error in learned weights cannot be computed for the complete models because they do not explicitly consider wind; thus, there are no "true" weights for comparison. The results demonstrate that minimizing error in one-step predictions yields much more useful models. They also demonstrate that, when minimizing one-step error, linear regression is more effective than evolution. Furthermore, linear regression requires vastly less computation time than evolution. This difference is not surprising since evolution requires on average approximately 2000 evaluations to evolve a model. By contrast, linear regression requires only one sweep through the flight data to estimate model weights.

5.2 Learning Wind Models

After the competition ended, the source code was released, revealing the exact helicopter dynamics and parameters. These dynamics are based on the representation developed by Abbeel et al., except for the addition of wind parameters in the first two equations:

$$\begin{aligned}
u_{t+1} - u_t &= C_u(u_t + wind_u) + g_u + w_u \\
v_{t+1} - v_t &= C_v(v_t + wind_v) + g_v + D_v + w_v
\end{aligned}$$

The competition simulator was modeled after one specific helicopter type, an XCell Tempest whose dynamics are well known. Thus, the parameters $\vec{C}$, $\vec{D}$, and $\vec{w}$ are known constants and only wind values change across SDPs in the training and test sets. These values are generalized as described in Section 3. Given this additional information, learning a model of a particular SDP reduces to estimating the generalized parameters, $wind_u$ and $wind_v$.

To learn such wind models, we consider the same three approaches used to learn complete models. The two evolutionary approaches work in the same way, though now only two weights need to be learned. Also, when evolving models to minimize error in reward, the fitness function is based on the average error across 10 policies, each optimized for a different SDP in the training set. We found that, unlike with the complete models, this fitness function resulted in better performance than using only the generic policy. For linear regression, estimating the wind parameters when the others are known reduces to solving the first two equations of the model with respect to wind:

$$\begin{aligned} wind_u &= \frac{u_{t+1} - u_t - g_u}{C_u} - u_t \\ wind_v &= \frac{v_{t+1} - v_t - g_v - D_v}{C_v} - v_t \end{aligned}$$

Since noise is modeled as zero-mean Gaussian random variables, the average noise will be zero and is thus omitted.

As with the complete models, we evaluated these approaches by using them to learn models on the test SDPs, using those models to evolve policies, and evaluating the best evolved policies over 5 episodes in the SDP on which the model was trained. The bottom part of Table 5 shows the results of these comparisons. Unlike in the complete model case, minimizing error in reward now performs as well as the other approaches in terms of both error and reward of evolved policies. However, the need to evaluate weight settings on 10 different policies leads to much higher computation time. Evolving weights that minimize one-step error now performs as well as linear regression but remains an order of magnitude slower.

6. MODEL-FREE VS. MODEL-BASED APPROACHES

In this section, we compare several strategies for tackling the full generalized helicopter hovering problem from the 2008 RL Competition. Each strategy uses the policy-evolving and model-learning components presented in Sections 4 and 5 in different ways. The goal of each strategy is to maximize the cumulative reward the agent accrues over all evaluation episodes: 1000 episodes on every different test SDP, each with different wind settings.

The robustness analysis presented in Section 4 shows that, while it is possible to evolve a policy that will not crash on an arbitrary SDP, such a policy will not perform as well as one optimized for that SDP. Thus, excelling in the competition requires learning on-line in order to adapt to each test SDP. At the same time, a good agent must avoid crashing the helicopter and must minimize the time spent evaluating suboptimal policies during learning.

Therefore, a naïve strategy of evolving a new policy online for each test SDP is impractical. As shown in Figure 2, tens of thousands of evaluations are required to evolve a strong policy. Even if evolution could find a good policy in 1000 episodes, it would accrue large negative reward along the way. As mentioned in Section 5, models of the environment learned from flight data can be used to reduce the sample complexity of on-line learning. However, at the time of the competition, we were unable to learn models accurate enough to serve as reliable fitness functions for evolution.

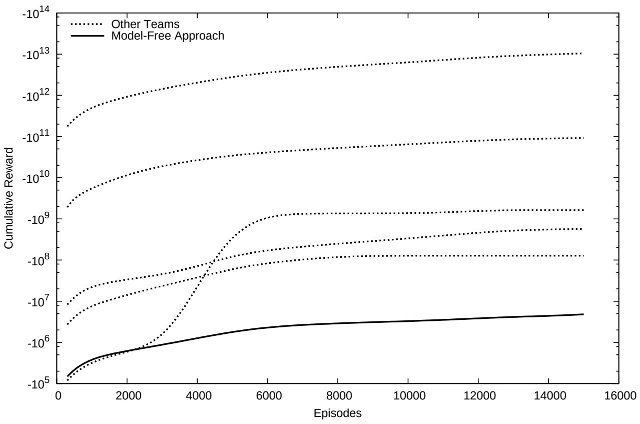

Figure 3: Cumulative reward accrued by competitors in the generalized helicopter hovering event of the 2008 RL Competition (lower is better). The test SDP changes every 1000 episodes.

Instead, we devised a simple, sample-efficient *model-free* approach. In advance of the competition, specialized policies for each of the 10 training SDPs were evolved using the procedure described in Section 4. Then, for each test SDP of the competition, the first 10 episodes were spent evaluating each of these specialized policies in that test SDP. Finally, whichever specialized policy performed the best was used for the remaining 990 episodes of that test SDP. This strategy allows the agent to adapt on-line to each test SDP in a sample-efficient way, without needing an accurate model.

Figure 3 shows the results of the generalized helicopter hovering event at the 2008 RL Competition, in which this model-free approach won first place. During the competition only 15 SDPs from the total set of 100 test SDPs were used for testing. Of the six teams that successfully completed test runs, only one other team managed to avoid ever crashing the helicopter, though it still incurred more negative reward. Another team matched the performance of the model-free approach for approximately the first third of the test run. However, three subsequent crashes relegated this team to a fourth place finish.

After the competition, we successfully implemented the complete model-learning algorithms described in Section 5.1 and tested a *model-based* approach to generalized helicopter hovering. Given some test SDP, one episode of flight data is gathered using the generic policy, which avoids crashing but may not achieve excellent reward on that SDP. Next, a complete model of the test SDP is learned from the flight data via linear regression, the best performing method. Then, neuroevolution is used to evolve a policy optimized for this SDP, using the model to compute the fitness function. Finally, the evolved policy controls the helicopter for all remaining episodes on that SDP.

We also tested an *incremental model-based* approach which works the same way but continues to evolve new policies using updated models as more flight data is gathered. Specifically, the incremental approach learns a new model at the end of each episode using all the flight data gathered on that SDP. Then, evolution is repeated using the latest model to find a new policy for the next episode. Once the performance of the policy in the SDP is at least as good as that

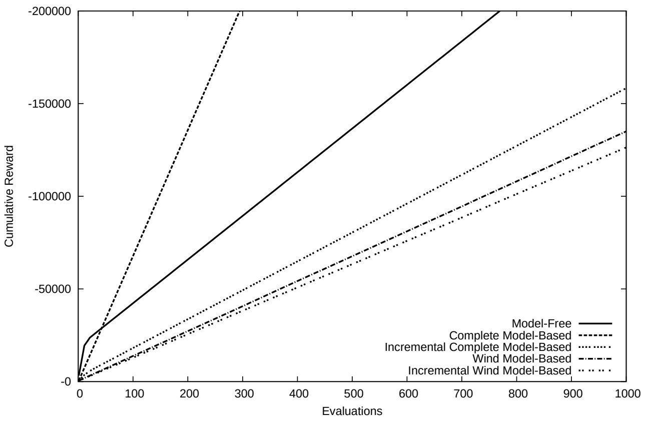

Figure 4: Cumulative reward averaged over 100 test SDPs of the model-free and model-based strategies for generalized helicopter hovering.

predicted by the model, learning is halted and that policy is employed for the remaining episodes. The incremental approach is similar to traditional methods for model-based RL (e.g., [20, 12]), but evolution, rather than dynamic programming [5], is used to find a policy given the model. It is also similar to apprenticeship learning [2], though the initial flight data is gathered by the generic policy instead of a human expert.

Both the model-based and incremental model-based strategies can also be implemented using the wind models described in Section 5.2. Unlike the complete models, which rely only on information publicly available during the competition, the wind models are learnable only given information gleaned from the post-competition software release. Hence model-based approaches relying on wind models would not have been feasible alternatives for the competition. Nonetheless, we tested their performance to isolate the challenge of generalizing across SDPs and to establish an upper bound on performance (i.e., a lower bound in the graphs) for the generalized helicopter hovering task.

To test these methods, we applied them to each of the 100 test SDPs and measured the average cumulative reward they accrued over 1000 episodes. Figure 4 shows the results. The model-free approach used in the competition gathers a lot of negative reward in the first 10 episodes as it evaluates each of the policies optimized for the training SDPs. Thereafter, its cumulative negative reward grows more slowly, as it uses only the best of these policies.

The model-based approach using a complete model performs worse than the model-free approach. Due to noise in the flight data, linear regression cannot always learn an accurate model given only one episode of flight data. Thus, the policies evolved using that model sometimes perform poorly in the corresponding test SDP. However, the incremental model-based approach using a complete model performs better than the model-free approach. By continually gathering flight data for learning, it reliably finds an accurate model within a few episodes. Given more knowledge about the helicopter dynamics, i.e., the equations shown in Section 5.2, it is possible to learn more accurate models. Hence, both model-based strategies using wind models outperform those using complete models, with the incremental version performing marginally better.

7. DISCUSSION

The results of our empirical analysis demonstrate that evolutionary computation is effective at optimizing the weights of neural network helicopter controllers. The superior performance of evolution when seeded with the baseline controller shows that evolution starting from scratch sometimes becomes trapped in local maxima, though such effects could potentially be reduced using *fitness sharing* [7] or other techniques for preventing premature convergence. While the neuroevolutionary methods we tested could not evolve topologies that outperformed the manually designed one, it is possible that other topology-evolving methods could do so. It is more likely, however, that there is little room for improvement in the manually designed topology, which was highly engineered by human experts.

Our results also show that models trained to minimize one-step error can produce more accurate fitness functions for policy evolution than those trained to minimize error in reward per episode. This is surprising since the latter approach directly optimizes the model for its true purpose: predicting a policy's reward. However, learning is easier in the former approach because it gets more training points from the flight data: one per timestep instead of per episode. Linear regression proves superior to evolution for optimizing model weights, yielding accurate models in much less time. However, in cases where a linear representation is inadequate or no human expert can provide one, evolution may prove a practical way to optimize model weights.

In addition, our results demonstrate the power of model-based approaches to maximize cumulative reward in the generalized problem, as these methods outperform even the competition-winning model-free approach. However, the poor performance of the (non-incremental) model-based approach using a complete model underscores the importance of learning a reliable model. Policies evolved with an inaccurate model perform worse than not using a model at all.

In comparing the model-free and model-based approaches, we measure only cumulative reward, as this was the sole criterion in the competition. However, in more realistic settings computational constraints may also be important. In such cases, the model-free approach is an appealing choice. The method is simple to implement and its performance, though inferior to that of the model-based based methods, is still quite strong. Furthermore, because the evolutionary runs are done off-line during training, only negligible computation time is needed during testing. By contrast, the model-based approaches are more complex to implement and involve conducting multiple evolutionary runs during testing.

Two characteristics of the generalized problem, as defined in the 2008 competition, are critical to the feasibility of the approaches we consider here. First, the fitness of a policy can be reliably estimated in a single episode. Stochasticity in the transition function makes the fitness function noisy, but in practice this noise is not large enough to necessitate *resampling* [4, 16], i.e., averaging fitness estimates over multiple episodes. If resampling were required, the model-free approach would need more episodes to determine which specialized policy to use in each test SDP, lengthening the period in which it accrues a lot of negative reward. Resampling would also greatly slow policy evolution, exacerbating the computational expense of the model-based approaches.

Second, there exist policies, e.g., the generic policy, that do not crash on any of the SDPs, regardless of the wind setting. This characteristic greatly reduces the danger of exploration. If such policies could not be discovered, the model-free approach would be impractical, as it assumes that each specialized policy, while potentially suboptimal, will not incur catastrophic negative reward. Similarly, the model-based approaches require a safe policy to gather flight data for model learning.

8. FUTURE WORK

In the future, we intend to study a more broadly generalized version of the helicopter hovering domain, one in which all of the model weights, not just those related to wind, can differ between SDPs. We hypothesize that doing so will eliminate the second characteristic mentioned above, as no single policy will be able to reliably avoid crashing in such disparate environments.

To tackle such a problem, we plan to develop evolutionary methods for finding policies that minimize the risk of crashing. One approach is to devise a fitness function based on cross-validation, thus explicitly measuring each policy's ability to generalize to previously unseen SDPs. Doing so can minimize the negative reward incurred while testing policies in the model-free approach or gathering flight data in the model-based approach. Since such policies might still occasionally crash or behave erratically, we may need to consider more elaborate model representations, as the one employed here is known to be unreliable outside the region in which the helicopter strives to hover [1]. The more broadly generalized problem may also require more elaborate network topologies that use recurrent links to mitigate the effects of noise, as such topologies have proven useful for helicopter maneuvers other than hovering [15].

9. ACKNOWLEDGEMENTS

We would like to thank Nate Kohl and Joe Reisinger for their helpful comments on a draft of this work.

10. REFERENCES

[1] P. Abbeel, V. Ganapathi, and A. Y. Ng. Learning vehicular dynamics with application to modeling helicopters. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2006.

[2] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, pages 1–8. ICML, 2005.

[3] J. Bagnell and J. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation 2001*. IEEE, May 2001.

[4] T. Beielstein and S. Markon. Threshold selection, hypothesis tests and DOE methods. In *2002 Congresss on Evolutionary Computation*, pages 777–782, 2002.

[5] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ., 1957.

[6] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1989.

[7] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.

[8] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. In *Proceedings of the European Conference on Machine Learning*, 2006.

[9] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, 1996.

[10] L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[11] R. Koppejan. Neuroevolutionary reinforcement learning for generalized helicopter control. Master's thesis, Universiteit van Amsterdam, 2009.

[12] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[13] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[14] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang. Inverted autonomous helicopter flight via reinforcement learning. In *Proceedings of the International Symposium on Experimental Robotics*, 2004.

[15] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*, 2004.

[16] P. Stagge. Averaging efficiently in the presence of noise. In *Parallel Problem Solving from Nature*, volume 5, pages 188–197, 1998.

[17] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[18] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning in robocup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[19] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[20] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224, 1990.

[21] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachussets, 1998.

[22] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[23] S. Whiteson, N. Kohl, R. Miikkulainen, and P. Stone. Evolving keepaway soccer players through task decomposition. *Machine Learning*, 59(1):5–30, 2005.

[24] S. Whiteson and P. Stone. Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, 7(May):877–917, 2006.

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