Evol. Intel. (2011) 4:219–241
DOI 10.1007/s12065-011-0066-z

RESEARCH PAPER

Neuroevolutionary reinforcement learning for generalized control of simulated helicopters

Rogier Koppejan · Shimon Whiteson

Received: 6 November 2010 / Accepted: 10 October 2011 / Published online: 30 October 2011

Abstract This article presents an extended case study in the application of neuroevolution to generalized simulated helicopter hovering, an important challenge problem for reinforcement learning. While neuroevolution is well suited to coping with the domain's complex transition dynamics and high-dimensional state and action spaces, the need to explore efficiently and learn on-line poses unusual challenges. We propose and evaluate several methods for three increasingly challenging variations of the task, including the method that won first place in the 2008 Reinforcement Learning Competition. The results demonstrate that (1) neuroevolution can be effective for complex on-line reinforcement learning tasks such as generalized helicopter hovering, (2) neuroevolution excels at finding effective helicopter hovering policies but not at learning helicopter models, (3) due to the difficulty of learning reliable models, model-based approaches to helicopter hovering are feasible only when domain expertise is available to aid the design of a suitable model representation and (4) recent advances in efficient resampling can enable neuroevolution to tackle more aggressively generalized reinforcement learning tasks.

Keywords Neural networks · Neuroevolution · Reinforcement learning · Helicopter control

R. Koppejan · S. Whiteson (✉)
Informatics Institute, University of Amsterdam,
Science Park 904, 1098 XH Amsterdam, The Netherlands
e-mail: s.a.whiteson@uva.nl

R. Koppejan
e-mail: r.koppejan@uva.nl

1 Introduction

The field of *reinforcement learning* (RL) [40, 82] aims to develop methods for solving sequential decision problems, typically formulated as Markov decision processes (MDPs) [9], wherein agents interact with unknown environments and seek behavioral policies that maximize their long-term reward. Developing effective RL methods is important because many challenging and realistic tasks (e.g., robot control [78], game-playing [86], and system optimization [89]) can be naturally cast in this framework.

In recent years, researchers have begun organizing RL Competitions [93] in an effort to stimulate the development of practical methods. To encourage robust methods, many competition events are formulated as *generalized tasks* [91]. A generalized task is not a single MDP but a distribution over related MDPs that vary along certain dimensions (e.g., sensor noise or environment size). A few training MDPs drawn from this distribution are available to competition participants in advance. However, the competition is decided by performance on test runs that use independently sampled MDPs. Since the agent does not know *a priori* which MDP it faces in each test run, it must learn *on-line* in order to maximize performance.[1]

One such generalized task is based on the problem of *helicopter hovering* [6], in which a 3-dimensional simulated helicopter strives to hover as close as possible to a fixed position. As in many other robotics applications, learning or evolving a control system for a helicopter is an appealing way to reduce the time and effort of manually

[1] The term 'on-line' has many meanings. In this article, we use it only to refer to the fact that the agent's performance is evaluated during learning. It is not meant to imply other constraints (e.g., scarce battery power or computational resources) that a deployed agent might face.

engineering a solution. As Bagnell and Schneider write, "Learning control entices us with the promise of obviating the need for the tedious development of complex first principle models and suggests a variety of methods for synthesizing controllers based on experience generated from real systems" [6]. Doing so is particularly useful in helicopter control since manually designing controllers is notoriously difficult due to their inherent instability, especially at low speeds [57]. As a result, helicopter control is a well-studied problem for which a wide range of solution approaches have been proposed, e.g., [34, 60, 66]. Numerous methods based on apprenticeship learning [1, 2, 3, 84] and policy-search reinforcement-learning [6, 49, 57], including evolutionary approaches [18, 19], have been developed.

However, the formulation of the helicopter control problem used in the competition poses two new challenges. First, the agent must explore a dangerous environment safely and efficiently in order to perform well. In previous work, learning relied on data gathered while a human expert controlled the helicopter. In contrast, no such expert is available in the competition task. Instead, the agent must explore its environment and thereby risk crashing the helicopter in order to gather such data. Second, the task is generalized such that wind velocities vary from one learning run to the next. Because wind greatly affects the helicopter's control dynamics, no single policy performs well across tasks. Consequently, the agent must learn on-line to find a policy specialized to each MDP it faces. Generalized helicopter hovering is thus an important challenge problem for RL, as it combines the difficulties of the original task (e.g., complex transition dynamics and high-dimensional state and action spaces) with the need to explore efficiently and learn on-line.

The problem also represents an interesting application domain for *neuroevolution* [72, 99], in which evolutionary methods [27] are used to optimize neural networks. On the one hand, such methods have proven effective in difficult RL tasks [28, 29, 43, 55, 76, 88, 89, 94]. In fact, since large state and action spaces can be troublesome for traditional temporal-difference methods [80], helicopter hovering may be particularly well suited to direct-optimization methods like neuroevolution. On the other hand, the task poses unusual challenges for neuroevolution. Most work in neuroevolutionary RL focuses on *off-line* tasks that assume access to an environment simulator in which the agent can safely try out candidate policies. In contrast, generalized helicopter hovering is an *on-line* task. The agent must learn by direct interaction with each test MDP, during which the rewards it receives count towards its final score. While some methods exist for applying neuroevolution to on-line tasks [14, 89, 90], they would still require an unrealistic number of *samples* (i.e., interactions with the environment) for a high-risk task like helicopter hovering.

This article presents an extended case study in the application of neuroevolution to generalized helicopter hovering. We propose several methods, examine the practical issues involved in their construction, and analyze the trade-offs and design choices that arise. In particular, we describe and evaluate methods for three versions of generalized helicopter hovering.[2]

First, we consider *2008 generalized helicopter hovering* (GHH-08), the task used in the 2008 RL Competition. We describe the simple model-free strategy with which we won first place in the competition. Furthermore, we describe a more complex model-based approach developed later and compare several ways of learning helicopter models. Finally, we present a post-competition analysis showing that, under some circumstances, the model-based approach can substantially outperform the model-free approach.

Second, we consider *2009 generalized helicopter hovering* (GHH-09), the task used in the 2009 RL Competition. In this variation, wind is generalized in a more complex way that renders unreliable the model-based methods that excel in GHH-08. To address this, we propose a hybrid approach that synthesizes model-free and model-based learning to minimize the risk of a catastrophic crash. We present the competition results, in which this method performed best in 19 of 20 test runs but crashed on the twentieth, earning a second place finish. We also present a post-competition analysis that isolates the cause of the crash and assesses whether the competition results are representative of the method's efficacy.

Finally, we propose a third variation on the task, which we call *fully generalized helicopter hovering* (FGHH). Our results on both GHH-08 and GHH-09 demonstrate that safe exploration is feasible in both variations. It is easy to find a *generic policy* that, while suboptimal, is robust enough to avoid crashing on any MDP. Such a policy can then be used to safely gather the flight data needed to learn a model. FGHH complicates exploration by generalizing the entire helicopter environment, rather than just wind velocities. As a result, it is infeasible to find a fully robust generic policy and computationally expensive to find one that minimizes the risk of crashing. To address this challenge, we propose and evaluate an extension to our model-based method that adapts recently developed methods for efficient *resampling* [32] to minimize the computational cost of discovering a good generic policy.

This article makes four main contributions. First, we demonstrate that neuroevolution can be an effective tool for complex, on-line reinforcement-learning tasks. While a

[2] Source code for all the tasks and methods described in this article is available at http://staff.science.uva.nl/~whiteson/helicopter.zip.

naïve approach has prohibitive sample costs, we propose and validate model-free and model-based methods that use neuroevolution as an off-line component to minimize the cost of on-line learning. Second, our results provide new evidence about the strengths and limitations of neuroevolution: while neuroevolution excels at learning the weights of fixed-topology neural networks, it does not discover effective topologies for helicopter policies and is less efficient at learning helicopter models than linear regression. Third, our results offer insight about the pros and cons of model-free versus model-based approaches in this setting. In particular, they demonstrate that, due to the difficulty of learning reliable models, model-based approaches to helicopter hovering are feasible only when a large amount of domain expertise is available to aid the design of a suitable model representation. However, given such a representation, they can greatly outperform model-free methods. Fourth, we demonstrate the potential of neuroevolution for solving aggressively generalized RL tasks such as FGHH. By exploiting the latest advances in efficient resampling, neuroevolution can effectively minimize the cost of exploration in such challenging tasks.

The rest of this paper is organized as follows. Section 2 provides background on MDPs, helicopter hovering, and generalized tasks. Sections 3, 4 and 5 describe, evaluate, and analyze methods for GHH-08, GHH-09, and FGHH, respectively.[3] Section 6 discusses the results, Sect. 7 outlines opportunities for future work, and Sect. 8 concludes.

2 Background

In this section, we give a brief overview of MDPs, describe the original helicopter hovering problem, and formalize the notion of generalized tasks.

2.1 Markov decision processes

The sequential decision problems addressed in RL are often formalized as MDPs [9], which can be described as 5-tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{S}$ is the set of all states the agent can encounter,
- $\mathcal{A}$ is the set of all actions available,
- $\mathcal{T}(s, a, s') = P(s'|s, a)$ is the transition function,
- $\mathcal{R}(s, a, s') = E(r|s, a, s')$, is the reward function, and
- $\gamma \in [0, 1]$ is the discount factor.

The process evolves over a series of discrete timesteps. In each timestep t, the agent observes its current state $s_t \in \mathcal{S}$ (e.g., via sensors on a robot) and selects an action $a_t \in \mathcal{A}$ (e.g., by activating motors on a robot). Following $\mathcal{T}$ and $\mathcal{R}$, the agent receives an immediate reward r_t and transitions to a new state s_{t+1}.

[3] The work presented in Sect. 3 appeared earlier in the form of a conference paper [47].

A policy $\pi(s, a) = P(a|s)$ indicates the probability of selecting each action in each state. An optimal policy $\pi^*(s, a)$ is one that maximizes the expected discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad (1)$$

For every MDP there exists at least one deterministic optimal policy $\pi^*(s)$ which directly maps states to actions [82]. In this article, we consider only deterministic policies.

When $\mathcal{T}$ and $\mathcal{R}$ are known to the agent, the result is a planning problem, which can be solved using *dynamic programming* methods [8]. When $\mathcal{T}$ and $\mathcal{R}$ are initially unknown, the agent faces a reinforcement learning problem. Such a problem can be solved using *temporal-difference* methods [80], which extend principles of dynamic programming to the learning setting. However, in many tasks, better performance can be obtained using a *policy search* approach [28, 41, 94], in which optimization methods such as evolutionary computation are used to directly search the space of policies for high performing solutions. This article focuses on neuroevolutionary policy search approaches.

2.2 Helicopter hovering

In the helicopter hovering task, an RL agent seeks a policy for controlling a 3-dimensional simulated helicopter. The dynamics of the task are based on data collected during actual flights of an 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 are complex. Second, both the state and action spaces are continuous and high dimensional. Third, the task involves high risk, as bad policies can crash the helicopter, incurring catastrophic negative reward. Here we give a brief description of the helicopter hovering problem; more details can be found in [6, 57].

A helicopter episode lasts 10 min. During this time, the helicopter simulator operates at 100 Hz, i.e., the helicopter state is updated 100 times per second, yielding 60,000 *simulator timesteps* per episode. However, helicopter control occurs at only 10 Hz, yielding 6,000 *control timesteps* per episode. Throughout this article, we refer to the current simulator timestep as *ts* and the current control timestep as *tc*.

Table 1 The 9-dimensional state vector

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)

Table 2 The 4-dimensional action vector

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

At each control timestep, the agent receives a 9-dimensional state vector[4] (see Table 1) and responds by specifying a 4-dimensional action (see Table 2). Although the transition function is stochastic (i.e., the agent does not know to what state it will transition before its action is complete), there is no noise in the agent's state signal (i.e., the state is fully observable and thus the Markov property holds). While this assumption would not strictly hold in a real helicopter, it is a reasonable approximation for a simulator, as modern systems often have access to sophisticated hardware and software for highly accurate state estimation. For example, several examples of control for real helicopters have relied on the well-known VICON motion-capture technology [19, 34, 60].

Together with the state vector, the agent also receives a negative immediate reward equal to 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. Since the target position is the origin, this is simply:

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

where s_i is the current value of the ith state feature. Reward is not discounted, i.e., $\gamma = 1$.

The helicopter begins each episode at the origin, i.e., $s_i = 0$ for all i. An episode ends prematurely if the helicopter crashes, which occurs if the velocity along any of the main axes exceeds 5 m/s, the position is off by more than 20 m, the angular velocity around any of the main axes exceeds 4 π rad/s, or the orientation is more than $\pi/6$ rad from the target orientation. Crashing results in a large reward penalty equal to the most negative reward achievable for the remaining time.

2.3 Generalized tasks

To discourage participants from overfitting to a single MDP and to encourage them to develop methods capable of robust, on-line learning, many events in the RL Competitions are formulated as generalized tasks [91, 92]. A generalized task $\mathcal{G} : \Theta \mapsto [0, 1]$ is simply a probability distribution over a set of tasks Θ. In the RL Competitions, $\mathcal{G}$ is not known to the participants. Instead, only a few MDPs sampled from this distribution are released for training. The competition is then decided by averaging the performance of each participant's agent across multiple test runs, with each run i conducted using a single task θ_i sampled independently from $\mathcal{G}$, where $\theta_i \in \Theta$ is an MDP.

At the beginning of each test run, the agent does not know which task has been sampled from $\mathcal{G}$. Except in degenerate cases, no fixed policy will perform well across Θ. Consequently, the agent must learn on-line during each test run in order to perform well in expectation across the test runs.

3 The 2008 generalized helicopter hovering task

The problem of generalized helicopter hovering was first introduced in the 2008 RL Competition. In this variation, which we refer to as GHH-08, wind was added to the helicopter simulator, significantly altering the transition dynamics. The details of how the task was generalized were kept secret during the competition. However, after the competition, the software, based on RL-Glue [85], was made public.[5] Each possible task θ is defined by two parameters:

- $wind_u \in [-5, 5]$, wind velocity in m/s in the x-axis, and
- $wind_v \in [-5, 5]$, wind velocity in m/s in the y-axis.

The probability distribution $\mathcal{G}$ is uniform over the set Θ of all possible values of this pair of parameters.

The presence of wind changes the way the helicopter responds to the agent's actions, in turn 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 MDP it faces and adapt its behavior accordingly.

[4] The simulator actually offers three additional state features describing angular velocities but we omit these from the state representation because they can be derived from the other nine state features.

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

Before the competition, 10 MDPs sampled from $\mathcal{G}$ were released for training. The competition itself was decided based on average performance across 15 runs, each consisting of 1,000 episodes on an MDP sampled independently from $\mathcal{G}$.

This section describes the solution methods we developed to tackle GHH-08. We first describe and evaluate, in Sects. 3.1 and 3.2, component methods that enable the agent to find policies and models for individual MDPs. Then, in Sects. 3.3 and 3.4, we describe and evaluate complete solution methods, one model-free and one model-based, for the generalized task.

3.1 Evolving helicopter policies

Before addressing the challenges of generalized helicopter hovering, we first consider how to evolve policies for a single helicopter MDP with fixed wind parameters. This process is a central component in each of the methods presented in Sects. 3.3 and 3.4 for tackling the full, generalized problem.

We use a neuroevolutionary approach, i.e., we evolve policies represented as neural networks. Given its numerous successes in difficult RL tasks [28, 29, 43, 55, 76, 88, 89, 94], especially those that are large, noisy, and continuous, neurevolution is well suited to this task. Other optimization methods, such as cross-entropy [17] or covariance matrix adaptation evolutionary strategies (CMA-ES) [30], could also be used. We do not include empirical comparisons against such alternatives in this article because our purpose is not to identify the best algorithm for optimizing helicopter policies. Instead, our goal is to investigate how to construct complete methods for generalized helicopter hovering. To this end, we employ neuroevolution as a representative optimization method for the subproblem of evolving policies.

In the helicopter problem, each neural network generated by evolution represents a different policy mapping states to actions. To find a good policy, we employ a simple *steady-state* neuroevolutionary method that builds its initial population off a given *prototype network*. The details of this algorithm, as well as the parameter settings used for all our experiments on GHH-08 and GHH-09 (i.e., those presented throughout Sects. 3 and 4), are specified in Appendix A.

The fitness function consists of the reward that the agent accrues during a single episode when using the policy specified by the network. Because of stochasticity in the transition function, the resulting fitness function is noisy. Therefore, we also tried using longer fitness evaluations in which reward is averaged over multiple episodes but found no performance improvement.

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* policy provided with the competition software. This baseline policy is robust in that it never causes the helicopter to crash. However, its performance is weak, as it is unable to consistently hover near the target point. This approach can be viewed as a simple form of *population seeding*, which has proven advantageous in numerous applications of evolutionary methods, e.g., [33, 38, 62].

In the third approach, we evolve *multi-layer perceptrons* (MLPs) using a topology manually constructed by human experts [57]. The topology, shown in Fig. 1, employs both sigmoid and linear activation functions.

In the fourth approach, we evolve MLPs but seed the initial weights such that it is equivalent to the baseline policy. To implement the linear baseline policy 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 state features described in Table 1, and four outputs, one for each action feature. In all four approaches, evolution optimizes only the weights of fixed-topology networks. However, we also tested two neuroevolutionary methods, NeuroEvolution of Augmenting Topologies (NEAT) [76] and Alternating Extension and Optimization of Neural Networks (AEONN) [46], 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

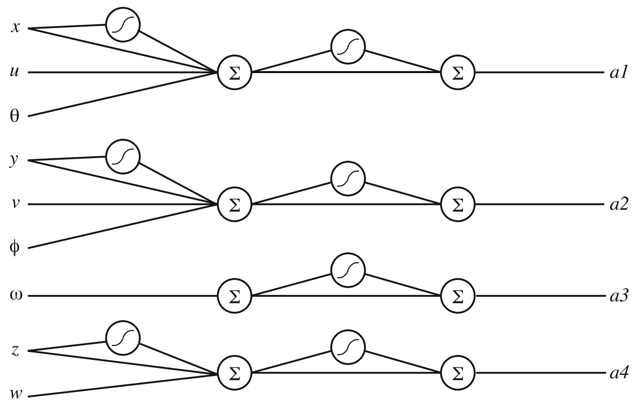

Fig. 1 The manually designed topology of a neural network helicopter policy, indicating which nodes use sigmoid activation functions and which use linear summations

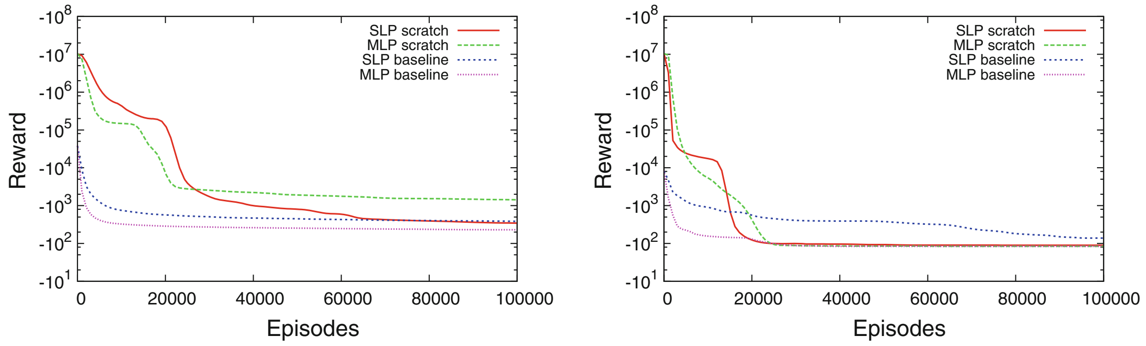

Fig. 2 Average (*left*) and best (*right*) performance of the population champion over time on GHH-08 (*lower* is better)

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 MDPs, i.e., 100 runs per approach and 400 runs total. The results, averaged over all 100 runs for each method, are shown in the left side of Fig. 2. Student's *t* tests confirmed that the differences in final performance between all approaches (except between the two SLP approaches, which converge to nearly identical scores) are statistically significant ($p < 0.0026$).

The results demonstrate that seeding the population with the baseline policy enables evolution to begin in a relatively fit region of the search space and therefore can significantly speed evolution. This is consistent with many earlier results confirming the efficacy of population seeding, e.g., [33, 38, 62]. The results also show that, when the baseline policy 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 population seeding. 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 policy, albeit more slowly (as in the best runs shown in the right side of Fig. 2). 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 MDP, with no attempt to generalize across MDPs with different wind settings. To determine the robustness of the resulting policies, we compared their average performance across all 10 training MDPs to their performance on the particular MDP for which they were trained. Specifically, we selected the best single-layer and multi-layer policy evolved for each MDP from the baseline policy and tested it for 10 episodes in every training MDP. The results are shown in Table 3.

Table 3 Performance of best SLP and MLP policies in GHH-08 evolved from the baseline policy: average reward (r) and standard deviation (σ) on the particular MDPs for which they were trained and average reward ($r_{\mathcal{G}}$) and standard deviation ($\sigma_{\mathcal{G}}$) across all training MDPs

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

This comparison demonstrates that the MLP policies are far more robust, achieving much better average performance and lower variance across the training MDPs. In fact, no specialized multi-layer policy crashes the helicopter on any of the 10 MDPs. By contrast, the single-layer policies frequently crash on MDPs 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 MDPs. This result underscores the challenges of achieving high performance in the generalized version of the task, which we address in Sects. 3.3 and 3.4.

3.2 Learning helicopter models

The results presented above demonstrate that neuroevolution can discover effective policies for helicopter hovering. However, doing so incurs high sample costs because 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, directly using evolution to find policies on-line is infeasible for the competition because participants are evaluated on the cumulative reward their agents accrue during learning. Even if methods for improving the on-line performance of neuroevolution [14, 89, 90] were used, such an approach would not be practical.

One way to reduce sample costs is to use a model-based approach. If the agent can learn a model of the environment from flight data for each testing MDP 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 costs. Doing so allows the agent to employ a good policy much earlier in the run, thus increasing its cumulative reward.

Learning such a model can be viewed as a form of *surrogate modeling*, also known as *fitness modeling* [36], in which a surrogate for the fitness function is learned and used for fitness evaluations. Surrogate modeling is useful for smoothing fitness landscapes [67, 98] or when there is no explicit fitness function, e.g., in interactive evolution [61, 69]. However, it is most commonly used to reduce computation costs [37, 59, 67, 68, 70] by finding a model that is faster than the original fitness function. The use of surrogate modeling in our setting is similar but the goal is to reduce sample costs, not computational costs.

Recall that, as described in Sect. 2.2, helicopter state is updated at 100 Hz while the agent observes this state and adjusts its action only at 10 Hz. Consequently, any learned model is necessarily an approximation, as it can only predict the next control timestep $tc + 1$ given the current control timestep tc and action, without explicitly modeling the simulator timesteps ts that occur in between. In practice, however, it is still possible to learn models that correctly predict how a given policy will perform in GHH-08.

During the competition, the details of the helicopter environment were hidden. However, helicopter dynamics have been well studied. In particular, Abbeel et al. [3] developed a representation of the transition function of a hovering helicopter that uses a set of linear equations to predict accelerations given a state and action at time ts.

$$\begin{aligned}
u_{ts+1} - u_{ts} &= C_u u_{ts} + g_u + D_u + w_u \\
v_{ts+1} - v_{ts} &= C_v v_{ts} + g_v + D_v + w_v \\
w_{ts+1} - w_{ts} &= C_w w_{ts} + g_w + C_{a_4} a_{4tc} + D_w + w_w \\
p_{ts+1} - p_{ts} &= C_p p_{ts} + C_{a_1} a_{1tc} + D_p + w_p \\
q_{ts+1} - q_{ts} &= C_q q_{ts} + C_{a_2} a_{2tc} + D_q + w_q \\
r_{ts+1} - r_{ts} &= C_r r_{ts} + C_{a_3} a_{3tc} + D_r + w_r
\end{aligned}$$

The accelerations depend on the values of the $\mathbf{g}$ vector, which represent gravity (9.81 m/s) expressed in the helicopter frame. The values of $\mathbf{w}$ 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 $ts + 1$. These velocities describe half the state at time $ts + 1$. The remaining half, which describes the helicopter's position, is estimated by adding the velocities at time ts to the position at time ts. As described in Sect. 2.2, reward is a simple function of the helicopter's current state, which can be approximated using the state 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 $\mathbf{C}$, $\mathbf{D}$, and $\mathbf{w}$.

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 used to evolve policies (see Appendix A). Fitness is based on the error in total estimated reward per episode using a single policy trained on an MDP 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 tc in an episode, of the state predicted at time $tc + 1$. 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 states and subtracting gravity $\mathbf{g}$ from the state at $tc + 1$. After preprocessing, evolution or linear regression is used to estimate $\mathbf{C}$ and $\mathbf{D}$. The noise parameters $\mathbf{w}$ are approximated using the average of the squared prediction errors of the learned model on the flight data.

We evaluated each of these approaches by using them to learn models for each of the 100 test MDPs that were released after the competition ended. Then we used the learned model to evolve policies in the manner described in Sect. 3.1 Finally, we took the best policy discovered in each evolutionary run and averaged its performance over 5 episodes in the MDP on which the corresponding model was trained.

The results, shown in Table 4, demonstrate that minimizing error in one-step predictions yields much more useful models. They also demonstrate that, when

Table 4 Performance of models learned via evolutionary computation to minimize error in reward (EC-MER) or in the next state (EC-MENO), or via linear regression (LR). Results compare 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 MDP for which the model was trained

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

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.

3.3 Model-free approach

In this section, we describe the simple model-free approach for tackling GHH-08 that won first place in the 2008 RL Competition.

The robustness analysis presented in Sect. 3.1 shows that, while it is possible to evolve a policy that will not crash on an arbitrary MDP, such a policy will not perform as well as one optimized for that MDP. Thus, excelling in the competition requires learning on-line in order to adapt to each test MDP. 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 on-line for each test MDP is impractical. Each competition test run lasts only 1,000 episodes but, as shown in Fig. 2, tens of thousands of episodes are required to evolve a strong policy. Even if evolution could find a good policy in 1,000 episodes, it would accrue large negative reward along the way. As mentioned in Sect. 3.2, 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.

Instead, we devised a simple, sample-efficient model-free approach. In advance of the competition, specialized policies for each of the 10 training MDPs were evolved using the procedure described in Sect. 3.1 Then, for each test MDP of the competition, the first 10 episodes were spent evaluating each of these specialized policies in that test MDP. Finally, whichever specialized policy performed the best was used for the remaining 990 episodes of that

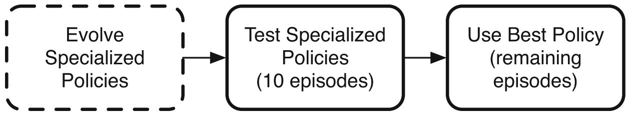
Evolve Specialized Policies
Test Specialized Policies (10 episodes)
Use Best Policy (remaining episodes)

Fig. 3 The model-free approach. The *dashed box* occurs *off-line* while the *solid boxes* occur *on-line*, during actual test episodes

test MDP. This strategy, depicted in Fig. 3, allows the agent to adapt on-line to each test MDP in a sample-efficient way, without needing an accurate model.

Figure 4 (left) shows the results of the generalized helicopter hovering event at the 2008 RL Competition, in which this model-free approach won first place. Of the six entries that successfully completed test runs, only one other entry managed to avoid ever crashing the helicopter, though it still incurred substantially more negative reward. In fact, all the competitors accrued at least two orders of magnitude more negative reward than our model-free approach. Due to these large differences, the results are shown in a log scale. Since this scale obscures details about the performance of the model-free method, the same results are also reproduced in a linear scale in Fig. 4 (right), showing how the slope rises or falls suddenly as the test MDP changes every 1000 episodes.

However, one entry matched the performance of the model-free approach for approximately the first third of testing. This entry, submitted by a team from the Complutense University of Madrid, also uses neuroevolution [51]. However, only single-layer feed-forward perceptrons are evolved. Furthermore, all evolution occurs on-line, using the actual test episodes as fitness evaluations. To minimize the chance of evaluating unsafe policies, their approach begins with the baseline policy and restricts the crossover and mutation operators to allow only very small changes to the policy. While their strategy makes on-line neuroevolution more feasible, three crashes still occurred during testing, relegating it to a fourth-place finish.

3.4 Model-based approach

After the competition, we successfully implemented the model-learning algorithms described in Sect. 3.2 and tested a model-based approach to generalized helicopter hovering, depicted in Fig. 5. Given some test MDP, one episode of flight data is gathered using the generic policy, which avoids crashing but may not achieve excellent reward on that MDP. Next, a complete model of the test MDP is learned from the flight data via linear regression, the best performing method. Then, neuroevolution is used to evolve a policy optimized for this MDP, using the model to compute the fitness function. Finally, the evolved policy controls the helicopter for all remaining episodes on that MDP.

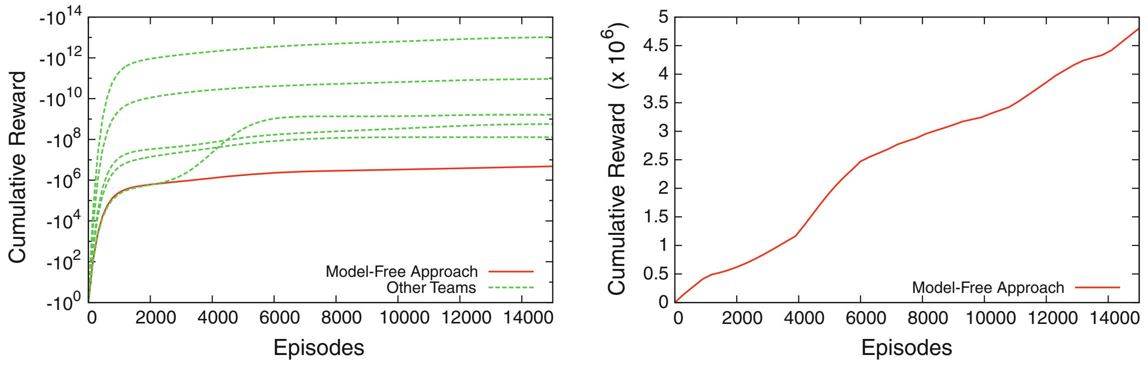

Fig. 4 At *left*, log-scale cumulative reward accrued by competitors in GHH-08 during the 2008 RL Competition (*lower* is better). At *right*, *linear-scale* cumulative reward of only the model-free approach in the same competition, showing how the slope rises or falls suddenly as the test MDP changes every 1,000 episodes

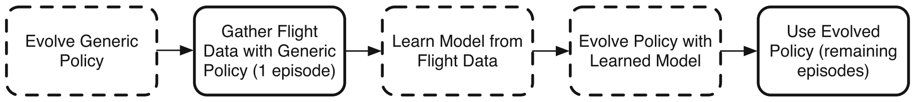
Evolve Generic Policy
Gather Flight Data with Generic Policy (1 episode)
Learn Model from Flight Data
Evolve Policy with Learned Model
Use Evolved Policy (remaining episodes)

Fig. 5 The model-based approach. The *dashed boxes* occur *off-line* while the *solid boxes* occur *on-line*, during actual test episodes

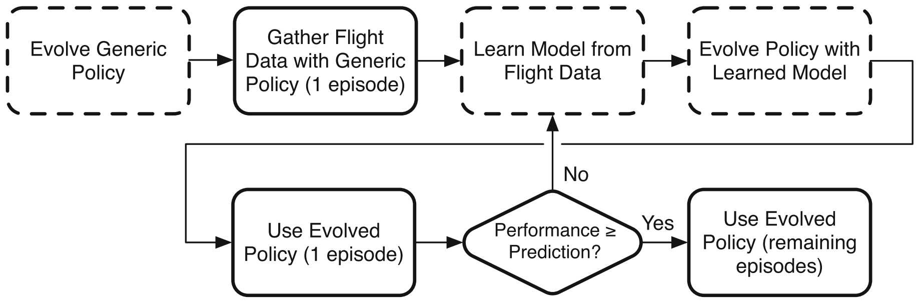
Evolve Generic Policy
Gather Flight Data with Generic Policy (1 episode)
Learn Model from Flight Data
Evolve Policy with Learned Model
No
Use Evolved Policy (1 episode)
Performance ≥ Prediction?
Yes
Use Evolved Policy (remaining episodes)

Fig. 6 The incremental model-based approach. The *dashed boxes* occur *off-line* while the *solid boxes* occur *on-line*, during actual test episodes

We also tested an incremental model-based approach, depicted in Fig. 6, 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 MDP. 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 MDP is at least as good as that 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., [54, 81]), but evolution, rather than dynamic programming [8], is used to find a policy given the model. It is also similar to apprenticeship learning [1–5], though the initial flight data is gathered by the generic policy instead of a human expert.

To test these methods, we applied them to each of the 100 test MDPs and measured the average cumulative reward they accrued over 1,000 episodes. Figure 7 shows the results. Student's t tests confirmed that the differences in final performance between the incremental model-based approach and the other approaches are statistically significant ($p < 0.0002$).

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 MDPs. Thereafter, its cumulative negative reward grows more slowly, as it uses only the best of these policies.

Surprisingly, the model-based approach 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 MDP. However, the incremental model-based approach performs better than the model-free approach. By continually gathering flight data for learning, it reliably finds an accurate model within a few episodes.

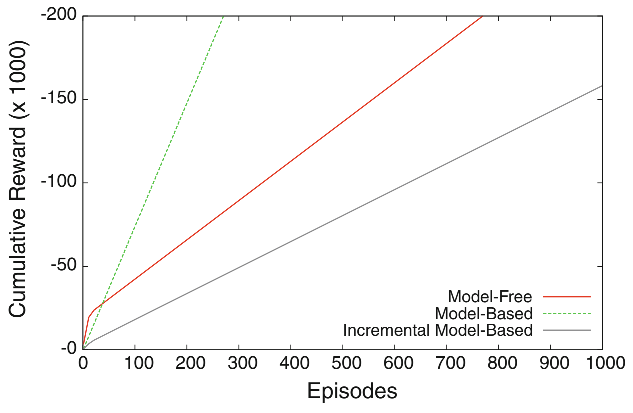

Fig. 7 Cumulative reward on GHH-08 of the model-free and model-based approaches, averaged over 100 test MDPs

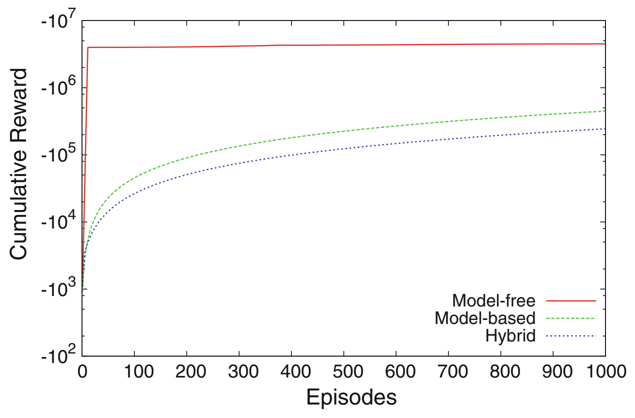

Fig. 8 Cumulative reward on GHH-09 of the model-free, incremental model-based, and hybrid approaches, averaged over 80 runs across 10 training MDPs

4 The 2009 generalized helicopter hovering task

For the 2009 Reinforcement Learning Competition, the generalized helicopter hovering task was modified to create a new challenge. The base task is the same and generalization is still restricted to variations in wind patterns. However, those wind patterns are considerably more complex. As in 2008, the details of the task were kept secret during the competition but were revealed afterwards.

In GHH-09, two sinusoidal wind currents run along the north-south and east-west axes. Each possible task θ is defined by four parameters describing these currents:

- $amp \in [-5, 5]$, maximum velocity,
- $freq \in [0, 20\pi]$, cycles per second,
- $phase \in [0, 2\pi]$, fraction of the wave period, and
- $center \in [0, 5]$, center amplitude of sine wave.

The probability distribution $\mathcal{G}$ is uniform over the set Θ of all possible values of these parameters.

In GHH-08, wind is fixed for every timestep in a given task. However, in GHH-09, wind changes over time. In fact, at each simulator timestep, the wind velocity for the north-south and east-west axes is computed as follows:

$$w_{ts+1} = amp \times \sin[freq \times (ts/100) + phase] + center$$

Setting *freq* and *phase* to zero yields a steady wind at *center*.

In the remainder of this section, we describe and evaluate the method we devised for GHH-09. In addition to the 2009 competition results, we present a post-competition analysis assessing whether the competition results are representative of our method's efficacy.

4.1 Hybrid approach

Due to the more complex wind dynamics, neither the model-free nor the model-based methods that excel in GHH-08 perform well in GHH-09. The model-free approach relies on the assumption that each specialized policy evolved off-line is robust enough to test on-line. In other words, while potentially suboptimal, these policies will not incur catastrophic negative reward. This assumption, which holds in GHH-08, does not hold in GHH-09.

To illustrate this effect, we conducted 80 runs testing the performance of the model-free method in GHH-09. Since no test MDPs were available before the competition, the training MDPs are used both for training and testing. First, a specialized policy was evolved for each of the 10 training MDPs. Then, 8 test runs were conducted with each of these MDPs. Since the agent does not know which MDP is used in each run, it tries out each specialized policy (except the one specifically trained for this MDP) during the initial episodes of each test run, as per the model-free method described in Sect. 3.3

The solid red line in Fig. 8 shows the resulting average cumulative performance. Though this method performs well on most runs, approximately 10% of the time the more complex wind dynamics lead to crashes when specialized policies evolved on one training MDP are tested on a different training MDP. Since even a single crash results in so much negative reward, the overall performance of the model-free method is poor.

Different problems arise for the incremental model-based approach. As described in Sect. 3.2, the model representation does not explicitly account for wind. This does not pose a problem for GHH-08, since wind velocities are approximately fixed in the helicopter frame. In GHH-09, however, wind velocity changes substantially over time. As a result, the incremental model-based approach cannot learn accurate models and thus cannot evolve high-performing policies off-line.

The dashed line in Fig. 8 illustrates this effect, showing the average cumulative performance of the incremental model-based method. As before, 80 runs were conducted, 8 on each training MDP. No crashes occurred during these runs, leading to better overall performance than the model-free method. A Student's *t* test confirmed that the final performance difference was statistically significant ($p = 5.8 \times 10^{-12}$). However, the policies produced by the incremental model-based method perform worse on average (−448.07 per episode) than the model-free method when it avoids crashing (−215.83 per episode).

An obvious solution would be to improve the model representation such that it explicitly accounts for wind velocity that changes over time. However, since the wind dynamics were not known during the competition, no prior knowledge was available that would guide the selection of a better model representation. We made several guesses, e.g., that stationary wind had been added to the z-axis, but were unable to find a representation that outperformed the one used for GHH-08.

Therefore, we designed a *hybrid approach* to try to minimize the weaknesses of both the model-free and incremental model-based approaches in GHH-09. We selected the two most reliable training MDPs[6] and evolved specialized policies for them. These MDPs are reliable in the sense that, in the experiments shown in Fig. 8, the specialized policies evolved for them never crashed on the other training MDPs. At the beginning of each test run, the hybrid method tests each of the two specialized policies by using it to control the agent for one episode on-line, as in the model-free approach.

Next, the flight data gathered during these two episodes is used to learn a model with linear regression, as in Sect. 3.4 Since evolution does not always produce good policies with such a model, the hybrid method conducts three independent evolutionary runs using this model and tests the resulting policies on-line for one episode each. Finally, the policy that performs best during these five initial episodes (two for policies generated by the model-free method and three for the incremental model-based method) is selected and used for the remainder of the test run.

4.2 Adding safeguards

While this hybrid approach tries to minimize the weaknesses of the model-free and incremental model-based approaches, it still allows some chance of a helicopter crash. In particular, thanks to model inaccuracies, it is possible that one of the three policies evolved from the model will crash when tested. It is also possible that one of these policies will perform well during its initial test but then crash on one or more episodes later in the run. While no crashes occurred during the evaluation of the incremental model-based method shown in Fig. 8, our informal experiments conducted leading up the 2009 competition showed that such crashes do occur approximately 0.5% of the time.

[6] MDP #4 and MDP #7 from the competition training set.

Since even one crash can devastate the overall performance of a helicopter agent, we augmented the hybrid method with safeguards designed to further reduce the chance of a helicopter crash. In particular, the agent tries to detect, within a single episode, whether the helicopter is headed towards a crash. If so, it replaces the current policy with the generic policy. As with GHH-08, the generic policy is evolved in the absence of wind and, while its performance is suboptimal, it never crashes the helicopter.

To detect when the helicopter is headed towards a crash, the agent checks at each control timestep *tc* whether the helicopter is in a dangerous state. We define a dangerous state as one in which position, velocity, or angular velocity along any axis falls outside the range [−0.5, 0.5]. By switching to the generic policy whenever a dangerous state occurs, the hybrid method further reduces the chance that any episode will result in a crash.

At the end of any episode in which a dangerous state occurs, the policy that induced that state is discarded. A new evolutionary run using the model is conducted to produce a replacement policy, which is used for the remainder of the test run or until it also induces a dangerous state. The hybrid approach is depicted in Fig. 9.

The blue dotted line in Fig. 8 shows the average cumulative performance of the hybrid method, including this safeguard. As before, 80 runs were conducted, 8 on each training MDP. Integrating the model-free and incremental model-based methods seems to improve performance over using either method alone. A Student's *t* test confirmed that the difference in final performance between the hybrid and model-free approaches is statistically significant ($p = 1.4 \times 10^{-14}$). However, due to high variance resulting from infrequent crashes by the incremental model-based method, the performance difference between it and the hybrid method was significant only at a 90% confidence level ($p = 0.07585$).

By testing only the safest specialized policies, the hybrid method strives to minimize the chance of a catastrophic

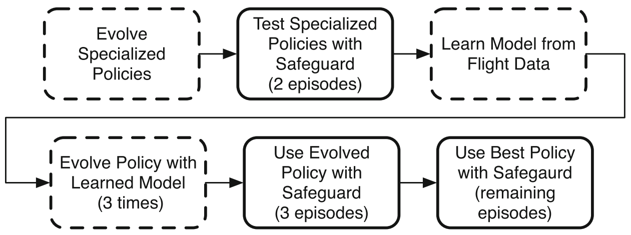
Evolve Specialized Policies
Test Specialized Policies with Safeguard (2 episodes)
Learn Model from Flight Data
Evolve Policy with Learned Model (3 times)
Use Evolved Policy with Safeguard (3 episodes)
Use Best Policy with Safegaurd (remaining episodes)

Fig. 9 The hybrid approach. The *dashed boxes* occur off-line while the *solid boxes* occur *on-line*, during actual test episodes

crash. By testing the results of several independent runs of evolution using the learned model, the hybrid method strives to maximize the chance that a high-performing policy will be selected for the remainder of each run. The safeguard also proved useful: on one run, the helicopter was headed towards a crash while using a policy evolved with the model. The safeguard switched to the generic policy, preventing a crash that would otherwise have erased the hybrid method's performance advantage.

While these results showed the safeguard's potential to improve performance, the results of the 2009 competition revealed limitations in its ability prevent crashes. Further experiments conducted after the competition showed that the success rate of the safeguard in preventing crashes depends critically on the choice of the fallback policy. In the following section, we present and discuss these additional results.

4.3 Competition and post-competition analysis

Based on its superior performance on the training MDPs, we submitted the hybrid method to the 2009 competition, the results of which are shown in Fig. 10. Three entries successfully completed the test runs but one incurred several orders of magnitude more negative reward than our entry (note the log scale in the figure). The other entry, an updated neuroevolutionary agent submitted by the team from the Complutense University of Madrid, was much more competitive but still accrued 85% more negative reward during the first 19 test runs. However, on the twentieth test run, our entry crashed the helicopter during the third episode, while testing one of the policies evolved from the model. During this episode, the agent detected that the helicopter entered a dangerous state and switched to the generic policy. However, the helicopter was already

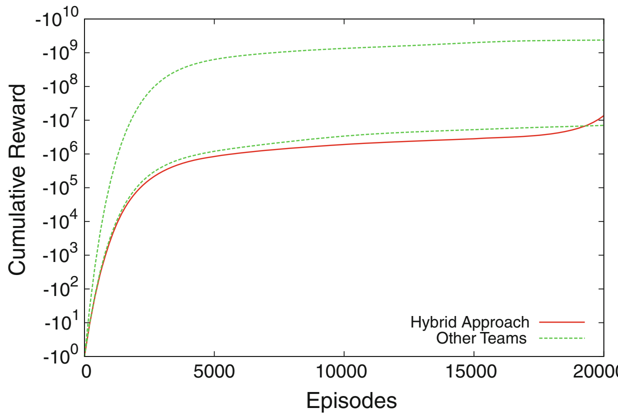

Fig. 10 Cumulative reward accrued by competitors in the generalized helicopter hovering event of the 2009 RL Competition (*lower* is better). The test MDP changes every 1,000 episodes

too close to crashing and the generic policy was unable to stabilize it. The negative reward incurred for this crash was great enough to relegate our entry to a second-place finish.

The crash that prevented the hybrid approach from winning was exactly the sort of event that the safeguards described above were designed to avert. Therefore, after the competition, we conducted additional experiments to analyze the cause of its failure and determine whether the competition was representative of the method's performance or merely bad luck.

To do so, we tested both the model-free and hybrid methods on the 20 MDPs used in the competition testing runs. For each MDP, we tested each method for 3 independent runs, for a total of 60 runs per method.

We hoped to conduct similar additional runs for the winning agent from the Madrid team. However, we were not able to reproduce the exact agent they submitted. Though they shared with us the source code they used for training, the behavior of their agent depends not only on this code but on the neural-network weights that resulted from an ad-hoc training regimen conducted before the competition. Unfortunately, they were not able to provide us with these weights or sufficient details of the training regimen to enable reliable reproduction of their agent (José Antonio Martin H., personal communication). However, we can still use the winning agent's performance during the competition as an unbiased estimator of its expected performance.

Figure 11 shows the average cumulative performance of the model-free and hybrid methods during these post-competiton runs, plotted against the average cumulative performance of the winning agent during the competition. As expected, the hybrid method reduces the chance of a crash: it crashed only 1% of the time, compared to 2.67% for the model-free method. As a result, it appears to modestly outperform the model-free method early on, though the difference is significant only at a 90% confidence level ($p = 0.08675$). However, because it falls back on the generic policy more often, it tends to accrue more negative reward per timestep when it does not crash, such that its final cumulative performance is only as good as that of the model-free method.

Furthermore, the hybrid method's cumulative performance is not as good as that of the winning agent, suggesting the outcome of the competition was not a fluke. However, the hybrid method was moderately unlucky in the competition: if crashes occur on 1% of runs, the probability of one or more crashes on 20 runs is only 18.21%. It is also possible that the winning agent was lucky and would occasionally crash if tested on all 100 MDPs. However, this seems unlikely, since the winning agent did not crash even on the difficult MDP that proved problematic for the hybrid method in the competition.

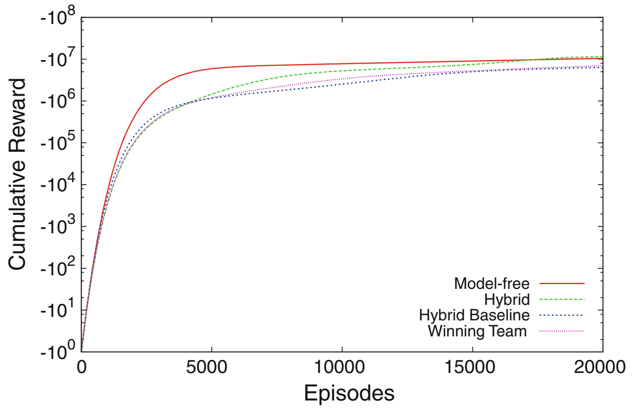

Fig. 11 Cumulative reward on GHH-09 of the model-free method, hybrid method, and hybrid method with the baseline policy as fallback instead of the generic policy, averaged over 3 runs on each of the 20 competition MDPs. The performance of the winning agent on the 20 competition runs is also *plotted* for comparison

In any case, the results make clear that the safeguards built into the hybrid method to minimize crashes are inadequate. From analyzing the runs that resulted in crashes, we concluded that the mechanism for determining that the helicopter is headed towards a crash works reliably, detecting a dangerous state well in advance of the crash. Therefore, we hypothesized that the problem lies in the generic policy to which the agent switches when such states occur. Though not designed to exploit a particular wind setting, the generic policy is still optimized to maximize performance (in the absence of wind). As a result, it may not be robust enough to save the helicopter when it is already in a dangerous state.

To test this hypothesis, we altered the hybrid method to fall back on the baseline policy rather than the generic policy. Recall from Sect. 3.1 that the baseline policy, provided with the competition software, is designed purely for robustness: it is not optimized for any MDP but merely tries to minimize the chance of crashing. We tested this method for 300 runs on the 100 test MDPs, the results of which are also shown in Fig. 11. Using the baseline policy improves the hybrid method's performance such that it approximately matches that of the winning agent. The improvement is a result of avoiding crashes; in fact, this method never crashed on any of the 300 test runs. However, due to high variance resulting from crashes by the original hybrid method, the performance difference between the two hybrid methods was significant only at a 90% confidence level ($p = 0.0849$).

Of course, avoiding crashes comes at a price, as the baseline policy performs worse than the greedy policy when the latter does not crash. We chose the generic policy as a fallback because, in training, it seemed to achieve this superior performance without increasing the chance of crashing (see Fig. 8). However, the post-competition analysis reveals that the training MDPs were misleadingly easy and the generic policy is not safe enough to use as a fallback. This design decision proved critical in the competition.

5 The fully generalized helicopter hovering task

While GHH-08 and GHH-09 both pose significant challenges, they also possess simplifying characteristics that are critical to the feasibility of the approaches described above. 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* [7, 74], 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 MDP, 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, such as the generic and baseline policies, that do not crash on any MDPs, regardless of the wind setting.[7] This characteristic greatly reduces the danger of exploration. In GHH-08, none of the specialized policies crash, making it safe to test each one online and thereby identify the strongest specialized policy for a given MDP. While not all the specialized policies are safe in GHH-09, enough are to make a model-free approach perform well. Without safe specialized policies, a model-free approach would be infeasible.

Similarly, the model-based approaches require a safe policy to gather flight data for model learning. For GHH-08, the model-based approaches use the generic policy to gather this data. For GHH-09, the hybrid methods use two safe specialized policies to gather flight data, though the generic policy is also safe. In lieu of such policies, the model-based approaches would not be able learn models without the risk of catastrophic negative reward.

To determine whether neuroevolutionary methods can be developed that tackle the helicopter hovering task without relying on these simplifying characteristics, we devised our own, more aggressively generalized version of the task, which we call *fully generalized helicopter hovering* (FGHH). The main idea is to make Θ, the set of possible MDPs, large enough that no single policy can reliably avoid crashing on MDPs drawn from $\mathcal{G}$.

[7] The generic policy never crashes when used for an entire episode but may, as noted in Sect. 4.3, crash when started in a dangerous state.

Table 5 Default model parameters for FGHH

C_u	−0.18	D_u	0.00	w_u	0.1941
C_v	−0.43	D_v	−0.54	w_v	0.2975
C_w	−0.49	D_w	−42.15	w_w	0.6058
C_p	−12.78	D_p	33.04	w_p	0.1508
C_q	−10.12	D_q	−33.32	w_q	0.2492
C_r	−8.16	D_r	70.54	w_r	0.0734

In GHH-08 and GHH-09, the basic transition dynamics are the same in every MDP and generalization occurs only across wind settings. In contrast, FGHH generalizes across the 12 parameters governing the transition dynamics (the C and D variables in the equations shown in Sect. 3.2). For simplicity, the noise parameters w are fixed and there is no wind. The distribution $\mathcal{G}$ over the resulting Θ is formed by setting a normal distribution over each parameter. The mean of this normal distribution is the default value for that parameter. These default values, shown in Table 5, correspond to the transition dynamics in GHH-08 and GHH-09 when there is no wind. The variance of the normal distribution is $(d\sigma)^2$, where d is the default value. Section 5.1 below describes how we set σ, which controls the breadth of generalization.

Generalizing the task in this way renders the model-free approach infeasible, as trying many specialized policies is too dangerous. Furthermore, it forces a model-based approach to directly address the challenges of exploration, since a generic policy must be found that minimizes the risk of crashing while gathering the flight data needed for model learning. Finally, it necessitates efficient methods for resampling, as accurately assessing any candidate generic policy requires averaging its performance over multiple MDPs sampled from $\mathcal{G}$.

The remainder of this section describes how we augment the incremental model-based approach to address these new challenges. Throughout, we assume full knowledge of $\mathcal{G}$. This is a practical necessity, since FGHH is not part of a competition and there are thus no organizers to select $\mathcal{G}$ and keep it hidden. Nonetheless, FGHH poses a real on-line learning challenge: though the agent knows $\mathcal{G}$, it does not know which θ sampled from $\mathcal{G}$ it faces on a given test run.

5.1 Fixed resampling approach

Given a suitably large value of σ, the generalization in FGHH ensures that no single policy will avoid crashing on any MDP drawn from $\mathcal{G}$. As a result, finding a suitable generic policy with which to gather flight data for model learning becomes more challenging. In GHH-08 and GHH-09, it was sufficient to evolve a generic policy in the absence of wind and simply assume it was reliable enough to safely gather flight data across all wind settings. As we show below, this approach fails in FGHH because a generic policy evolved on the default settings shown in Table 5 will often crash on other MDPs drawn from $\mathcal{G}$.

To address this difficulty, we augment the incremental model-based approach to more rigorously select a generic policy that minimizes the chance of crashing during flight data collection. In particular, a generic policy is evolved, not with a single MDP for the fitness evaluation, but rather with a new MDP sampled from $\mathcal{G}$ each time. When this evolutionary run completes, the best performing policy is used as a generic policy for the incremental model-based approach shown in Fig. 6. That is, in each test run, the generic policy is used to gather flight data, which is used to evolve a policy specialized to that MDP.

Sampling MDPs from $\mathcal{G}$ obviously introduces an enormous amount of noise into the fitness function. There are many possible ways to address this, e.g., by enlarging the population according to population-sizing equations that consider the variance of the fitness function [16, 25, 26, 31]. In this article, we focus on the well-known technique of *resampling*, i.e., averaging performance across multiple episodes. In this section, we consider a simple *fixed resampling approach*, in which each generic policy is evaluated for k episodes. In Sect. 5.2, we propose a more sophisticated resampling technique.

The fixed resampling approach evolves generic policies using a slightly different neuroevolutionary approach than that used for GHH-08 and GHH-09. In particular, it uses a generational algorithm rather than a steady-state one. This change was made to create consistency with the selection races approach (introduced below in Sect. 5.2), which requires a generational approach. The details of the generational algorithms and parameter settings are specified in Appendix B.

To fully specify FGHH, we must select a value for σ. Choosing σ properly is critical to creating a suitable task. If σ is too low, the task will not be sufficiently generalized and may not be any more difficult than GHH-08 or GHH-09. If σ is too high, the task will be too difficult, as even a resampling approach will not be able to find a generic policy stable enough to gather the flight data needed for model learning.

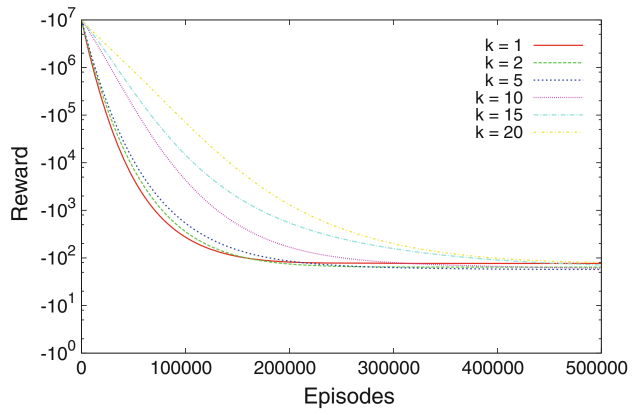

Fig. 12 Average performance of the population champion over time using fixed resampling with k episodes per evaluation on FGHH with $\sigma = 0.2$

To ensure an appropriate choice, we tested the fixed resampling approach at different values of σ and k to find a value of σ where the best performing value of k is greater than 1, indicating a need for resampling. We found that when $\sigma < 0.2$, no resampling is required. Fig. 12 shows the performance of the generation champion, averaged over 24 independent runs, for different values of k with $\sigma = 0.2$. Not surprisingly, there is a trade-off between the speed and quality of learning. Lower values of k make faster progress at the beginning because they can evaluate policies more quickly; higher values of k plateau higher because they can more accurately make fine distinctions between policies. While values of $k > 1$ perform best in the long run, good performance is still possible with $k = 1$. In contrast, Fig. 13 shows results when $\sigma = 0.3$. In this case, $k = 1$ performs poorly, as a single fitness evaluation is not enough to guide evolution. When $k = 2$, evolution makes significant progress but plateaus early. To achieve good performance, $k \geq 5$ is required.

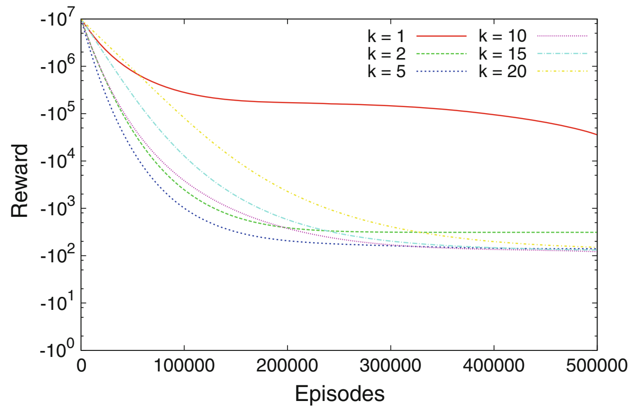

Fig. 13 Average performance of the population champion over time using fixed resampling with k episodes per evaluation on FGHH with $\sigma = 0.3$

5.2 Selection races approach

The results in the previous section make clear that a significant amount of resampling is required to evolve a good generic policy for FGHH. Consequently, the computational costs of the fixed resampling approach are much higher than those of the model-based approaches used for GHH-08 and GHH-09, since computing each fitness estimate requires simulating multiple episodes of evaluation. The increased computational costs make the fixed resampling approach less practical for a real-world setting. For example, when computational resources are limited, it will be necessary to terminate evolution early, before performance plateaus. As a result, flight data for model learning will be gathered using an inferior generic policy that accrues more negative reward and has a higher chance of crashing the helicopter.

To address this problem, we developed a second method for FGHH that we call the *selection races approach*. Like the fixed resampling approach, this method evolves a generic policy that minimizes the probability of crashing during flight data collection. As before, it evaluates each candidate policy on multiple MDPs sampled from $\mathcal{G}$. However, the number of episodes spent evaluating each policy is not fixed. Instead, episodes are allocated to policies dynamically, in an attempt to most efficiently determine which policies in the current generation should be selected as parents for the next generation.

Many strategies for making resampling more efficient have been developed, e.g., [7, 11, 12, 42, 56, 74]. For this work, we adapt a strategy recently proposed by Heidrich-Meisner and Igel [32] that uses *selection races*, a technique originally developed by Maron and Moore [50], to determine how many episodes to spend evaluating each policy. The main idea is to maintain confidence intervals around the fitness estimates of each candidate policy. These intervals are used to determine when, with probability δ, the best μ of the λ candidate policies in each generation have been selected for reproduction.[8]

In particular, policies are sorted into three pools: selected, discarded, and undecided. In each iteration, all the undecided policies are evaluated for one additional episode. If, as a result of these evaluations, the lower confidence bound of an undecided policy becomes larger than the upper confidence bound of at least $\lambda - \mu$ other policies, it is moved to the selected pool. Conversely, if the upper confidence bound of an undecided policy is lower than the lower confidence bound of at least μ other policies, it is

[8] Note that, while [90] also proposes an approach based on confidence intervals, it is not suitable here because it aims to maximize the reward accrued during on-line evolution as opposed to minimizing the computational cost of off-line evolution.

moved to the discarded pool. This process repeats until the selected pool contains μ policies or every undecided policy has already been evaluated for t_{limit} episodes. In either case, the μ policies with the highest estimated fitness at the end of the generation are selected for reproduction.

In principle, this method could be directly applied to the problem of selecting generic policies in FGHH. However, in practice, we had to substantially modify the method to produce an algorithm suitable to our setting. The first change concerns the way confidence bounds are computed. Both Heidrich-Meisner and Igel and Maron and Moore propose using the Hoeffding bound, which states that with probability δ the expected fitness of policy i is

$$X_i = \hat{X}_i \pm c_{i,t}$$

after t episodes of evaluation. Here $\hat{X}_i = \frac{1}{t}\sum_{j=1}^{t} X_{i,j}$, where $X_{i,j}$ is the fitness estimate from the jth evaluation of policy i. Furthermore,

$$c_{i,t} = (a-b)\sqrt{\frac{log(2/(1-\delta))}{2t}}$$

where $X_{i,j} \in [a, b]$.

The Hoeffding bound is problematic in the helicopter task because of the range of possible fitness estimates a policy can accrue on a given episode. A perfect policy will never accrue negative reward so $a = 0$. However, the worst possible policy crashes immediately and accrues the maximal possible negative reward on each timestep, yielding $b \simeq -1 \times 10^7$. Consequently, the confidence bounds are enormous. This would not be a problem if the typical difference in fitness between two policies was in a similar range. However, once evolution discovers policies that rarely crash, differences in fitness are typically many orders of magnitude less than $a - b$. As a result, using the Hoeffding bound is infeasible, as the conditions required to select or discard policies are never met.

Instead, we use Bayesian confidence bounds, as also proposed by Maron and Moore. This approach typically produces tighter bounds but assumes that the distribution over fitness values for each policy is normal. While this is not necessarily true in FGHH, the Bayesian confidence bounds still perform well in practice. As with the Hoeffding bound, these bounds state that with probability δ the expected fitness of policy i is $X_i = \hat{X}_i \pm c_{i,t}$. However, in our implementation, $c_{i,t}$ is now defined as:

$$c_{i,t} = Z_\delta \hat{\sigma}$$

where Z_δ is the number of standard deviations from the mean of a standard normal distribution required to contain area of size δ and $\hat{\sigma}$ is the estimated standard error. This estimate is calculated using the Bayesian method, which employs the Jeffreys prior and assumes that each of the sampled fitness values have the same mean and variance.[9]

The second change concerns the way the confidence bounds are updated. The algorithm described by Heidrich-Meisner and Igel keeps running estimates of these bounds and updates them only when new data makes them tighter. That is,

$$LB_i \leftarrow max\{LB_i, \hat{X}_i - c_{i,t}\}$$
$$UB_i \leftarrow min\{UB_i, \hat{X}_i + c_{i,t}\}$$

where LB_i and UB_i are the lower and upper confidence bounds of policy i, respectively. This approach works well in their experiments but leads to difficulties in the helicopter task, again because of the large range of possible cumulative rewards. The left side of Fig. 14 illustrates how a sequence of episodes evaluating a given policy can lead to a problematic scenario. In the first two episodes, the policy performs similarly but in the third episode it performs significantly worse. Since the new lower bound would be lower than the old one, it is not updated. As a result $\hat{X}_i < LB_i$. In other words, the mean is below the lower bound! This can lead to undesirable behavior in selection races. For example, the policy may be selected even if its expected fitness is low, because its lower bound will stay high no matter how far the mean drops.

To address this problem, we alter the update of the confidence bounds to incorporate new data regardless of whether the new resulting bounds are tighter:

$$LB_i \leftarrow \hat{X}_i - c_{i,t}$$
$$UB_i \leftarrow \hat{X}_i + c_{i,t}.$$

The right side of Fig. 14 shows the bounds updates that occur on the same example using this new scheme. Since the bounds are always updated after each episode, the mean stays within the lower and upper bounds and the weaker performance in the third episode produces a corresponding drop in the lower bound.

Algorithm 1 contains pseudocode of our modified selection races algorithms for picking μ policies in a given generation. First, each policy is given an initial evaluation and bounds are initialized (lines 5–8). In each iteration, each undecided policy receives another evaluation and the corresponding bounds are updated (lines 12–17). Then, policies are selected or discarded if appropriate (lines 18–25). Finally, if the selected pool is still too small once t_{limit} is reached, the highest ranked undecided policies are added to it (lines 34–39).

To evaluate this method, we conducted 24 independent runs in FGHH with $\sigma = 0.3$ at each of several values of δ.

[9] Our implementation uses the bayes_mvs method of the SciPy Python package.

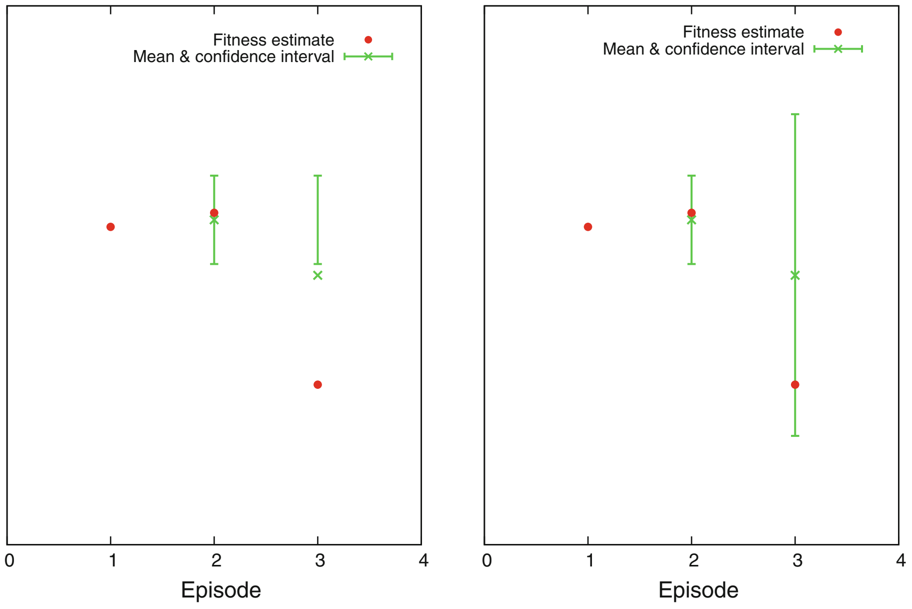

Fig. 14 At *left*, an example of unwanted behavior when always trying to tighten the confidence bounds as much as possible. With a confidence level of 90%, the average fitness drops below the lower bound at the 3rd evaluation. At *right*, the same scenario under our alternative scheme, which ensure that the mean stays between the *upper* and *lower* bounds

The other parameters were set as follows: $\lambda = 50, \mu = 20, t_{limit} = t_{min} = t_{max} = 10, \alpha = 0.1$. Since FGHH is so noisy, we found that setting $t_{limit} < t_{max}$ was pointless, since t_{limit} always quickly grew to reach t_{max}. Hence, we set $t_{limit} = t_{max}$, which effectively renders t_{min} and α irrelevant. The other neuroevolutionary parameters were set the same as with fixed resampling, as shown in Appendix B.

Figure 15 shows the resulting performance at each value of δ, compared to the fixed-resampling method at $k = 5$, the best-performing setting. At every setting, the selection races approach outperforms fixed resampling. Student's *t* tests confirmed that these performance differences are statistically significant ($p < 1 \times 10^{-4}$). By more efficiently allocating episodes of evaluation to candidate policies, the selection races approach is able to find better policies substantially more quickly.

Both these results and those presented in Sect. 5.1 measure only the performance of evolution while searching for generic policies. We also evaluated the performance of the incremental model-based approach in FGHH when using the best policies produced by evolution to gather flight data for learning models. In other words, we tested the strategy shown in Fig. 6 with either the fixed resampling or selection races method used to evolve the generic policy.

Figure 16 shows the results of 100 runs conducted for each method. In each run, the incremental model-based method used the generic policy produced from a different run of either fixed resampling or selection races, at the best-performing settings. As a baseline, the graph also shows the performance of the incremental model-based method using a generic policy discovered as was done for GHH-08 and GHH-09: by evolving it for a fixed MDP, without explicitly selecting for robustness across many MDPs. We used an MDP we call the *zero setting*, which corresponds to the mean of $\mathcal{G}$ for FGHH, i.e., using the default parameter values shown in Table 5.

The results verify the importance of evolving a generic policy that is sufficiently robust. Since FGHH is more aggressively generalized, generic policies evolved only for the zero-setting are not reliable, resulting in poor performance for the incremental model-based method. The results also underscore the importance of efficient resampling. Though fixed resampling explicitly searches for robust generic policies, wasteful evaluations slow it down. Consequently, the best generic policy found after 500,000 evaluations leads to performance for the incremental model-based method that is only marginally better than with a generic policy evolved just for the zero setting (the difference is not statistically significant). While fixed resampling could in principle discover better generic policies if run longer, the computational expense of doing so quickly becomes prohibitive. In contrast, thanks to smarter resampling, selection races are able to discover stronger generic policies in the same number of evaluations, yielding much better performance for the incremental model-based method. Student's *t* tests confirmed that the differences in final performance between the selection-races and zero-setting methods is statistically significant ($p = 0.0402$). Due to high variance resulting from infrequent crashes by the fixed-resampling method, the performance difference between it and selection races was significant only at a 90% confidence level ($p = 0.0706$).

Algorithm 1 Select $(\{x_1, \ldots, x_\lambda\}, \mu, t_{limit}, t_{min}, t_{max}, \alpha, \delta)$

```
1: $\mathbb{S} = \emptyset$ // selected individuals
2: $\mathbb{D} = \emptyset$ // discarded individuals
3: $\mathbb{U} = \{x_i \mid i = 1, \ldots, \lambda\}$ // undecided individuals
4: $t \leftarrow 1$ // current iteration
5: for all $x_i \in \mathbb{U}$ do
6:     $X_{i,t} \leftarrow evaluate(x_i)$ // initial evaluation
7:     $LB_i \leftarrow 0, UB_i \leftarrow 0$ // initial lower and upper bounds
8: end for
9: while $t < t_{limit} \wedge |\mathbb{S}| < \mu$ do
10:     $t \leftarrow t + 1$
11:     // reevaluate undecided policies
12:     for all $x_i \in \mathbb{U}$ do
13:         $X_{i,t} \leftarrow evaluate(x_i)$
14:         $\hat{X}_i \leftarrow \frac{1}{t}\sum_{j=1}^{t} X_{i,j}$
15:         // update $LB_i$ and $UB_i$ using Bayesian confidence bounds $c_{i,t}$
16:         $LB_i \leftarrow \hat{X}_i - c_{i,t}, UB_i \leftarrow \hat{X}_i + c_{i,t}$
17:     end for
18:     for all $x_i \in \mathbb{U}$ do
19:         if $|\{x_j \in \mathbb{U} \mid LB_i > UB_j\}| \geq \lambda - \mu - |\mathbb{D}|$ then
20:             $\mathbb{S} \leftarrow \mathbb{S} \cup x_i$ // select
21:             $\mathbb{U} \leftarrow \mathbb{U} \setminus x_i$
22:         else if $|\{x_j \in \mathbb{U} \mid LB_i < UB_j\}| \geq \mu - |\mathbb{S}|$ then
23:             $\mathbb{D} \leftarrow \mathbb{D} \cup x_i$ // discard
24:             $\mathbb{U} \leftarrow \mathbb{U} \setminus x_i$
25:         end if
26:     end for
27: end while
28: // update $t_{limit}$ depending on $|\mathbb{S}|$
29: if $|\mathbb{S}| = \mu$ then
30:     $t_{limit} = max(t_{min}, \frac{1}{\alpha} t_{limit})$
31: else
32:     $t_{limit} = min(t_{max}, \alpha\ t_{limit})$
33: end if
34:     // select best undecided policies if $\mathbb{S}$ is not full
35: while $|\mathbb{S}| < \mu$ do
36:     $x_i \leftarrow \arg max_{x_j \in \mathbb{U}} \hat{X}_j$
37:     $\mathbb{S} \leftarrow \mathbb{S} \cup x_i$
38:     $\mathbb{U} \leftarrow \mathbb{U} \setminus x_i$
39: end while
40: return $\mathbb{S}$
```

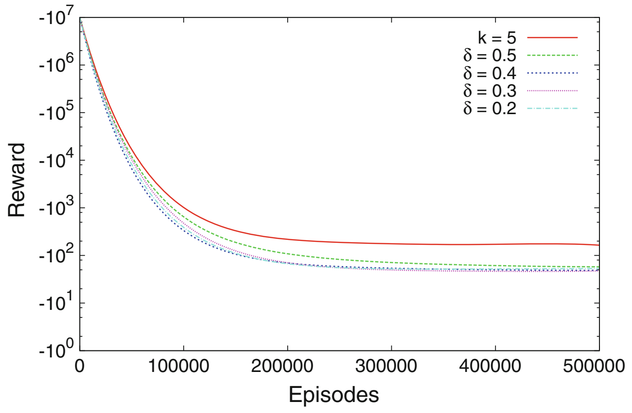

Fig. 15 Average performance of the population champion over time using selection races with confidence δ on FGHH with $\sigma = 0.3$, compared to the best-performing setting of fixed resampling

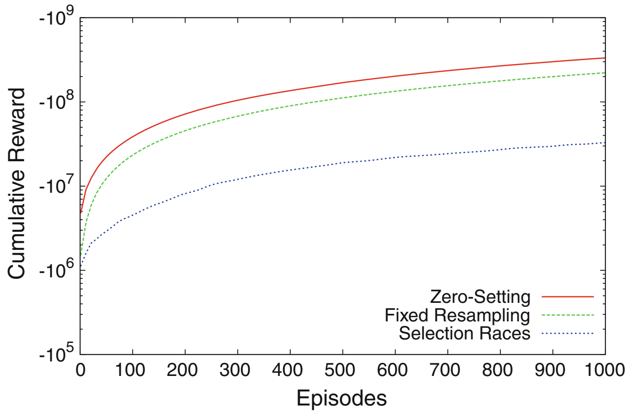

Fig. 16 Cumulative reward on FGHH with $\sigma = 0.3$ of the incremental model-based method using generic policies evolved for the zero-setting, using fixed resampling, or using selection races, averaged over 100 MDPs

6 Discussion

The experimental results presented in the preceding sections provide considerable evidence regarding the pros and cons of various neuroevolutionary approaches to generalized helicopter hovering. Strictly speaking, the results do not support any claims about how such methods might perform in other domains. This is especially the case since, by design, these methods were optimized for the specific challenges and constraints of these benchmark problems. However, we believe that these results nonetheless underscore some broader issues in both evolutionary computation and reinforcement learning.

First, the results highlight the importance of on-line learning for evolutionary computation. Evolutionary approaches have seen enormous success in reinforcement learning but the results often focus on off-line settings. While evolution has also succeeded on-line [58, 71, 77], especially in evolutionary robotics [20, 21, 45, 52, 64, 100], and some research has investigated customizing such methods to on-line settings [14, 15, 21, 65, 83, 89, 90], the problems tackled by evolutionary methods typically assume the availability of a fitness function that requires only computational resources to employ. Even in research on learning classifier systems [13, 35, 48, 73, 96, 97], some of which are closely related to traditional approaches in reinforcement learning, experiments commonly evaluate only the speed of learning and the quality of the resulting policy.

However, many reinforcement learning problems are on-line, in which case the amount of reward accrued during learning is more important. In such cases, the agent does not have an *a priori* model of its environment and evaluating a policy's fitness requires testing it in the real world or learning a model to use as a fitness function, both of which incur sample costs in addition to computational costs.

The results presented here illustrate how evolutionary methods can also excel in a challenging on-line setting. Rather than constituting a complete solution, evolutionary methods can serve as a policy-evolving component of a larger model-free or model-based solution. However, the results also reveal limitations in evolution's usefulness. While it proves an excellent means of discovering high-performing policies, it seems less effective in other roles such as learning helicopter models. In the particular setting examined in this article, in which domain knowledge allows the manual construction of a suitable model representation, simpler supervised methods such as linear regression appear better suited to this aspect of the task.

Second, the results illustrate how expensive exploration in reinforcement learning in realistic problems can be. Helicopter hovering is typical of many tasks (e.g., those involving physical robots and/or interacting with humans; high-risk tasks in security, surveillance, or rescue; and financial settings such as bidding or trading) in which a single erroneous action can have disastrous consequences. Classical approaches to exploration in reinforcement learning, such as ϵ-greedy [87], softmax [82], and interval estimation [39] all involve selecting each action in each state multiple times to assess its value, which is out of the question in such tasks. Even the most sophisticated methods, e.g., [10, 44, 79], which require only a polynomial number of samples to discover an approximately correct policy with high probability, explore much too liberally to be feasible in high-risk problems. Though an agent can in principle always compute a Bayes-optimal strategy for exploration [63, 95], doing so is typically computationally intractable. Thus, the dangerous nature of tasks such as generalized helicopter hovering underscores the need for heuristic methods that explore more conservatively.

The importance of efficient exploration suggests that model-based methods may have a significant advantage, as they can reuse samples in a way that reduces the need for exploration. The results in this article provide examples of that advantage, since the incremental model-based methods for GHH-08 greatly outperform even the competition-winning model-free approach. However, the results also demonstrate the limitations of model-based methods. For various reasons, including the fact that state is only observed at 10 Hz, learning a helicopter model that accurately captures the complex dynamics seems feasible only with extensive domain knowledge. Even slight inadequacies in the model representation, such as those used in GHH-09, can render the entire model-based approach unreliable.

Finally, the results also shed light on the challenges of designing suitable generalized tasks for use in reinforcement learning competitions and to serve as community benchmarks. While generalized tasks are an effective way to assess the robustness of on-line reinforcement learning methods [91, 93], selecting the right distribution over tasks can be tricky. Despite the best efforts of competition designers, GHH-08 proved insufficiently generalized. Previous work on helicopter hovering relied on data gathered by a human expert, thereby obviating the need for exploration. GHH-08 requires exploration but meeting this challenge is straightforward, since it is easy to find a generic policy with which to safely gather flight data and a single, fixed model representation suffices for all tasks in the distribution. While GHH-09 complicates the design of a model representation and makes exploration more dangerous, it is still possible to find safe policies for exploration and to evaluate a candidate policy in a single episode. FGHH addresses these shortcomings by making all exploration policies inherently risky and necessitating resampling when evaluating policies. However, selecting the right distribution for FGHH was possible only after extensive experiments with the fixed resampling approach.

7 Future work

Several directions for future research follow naturally from the work presented in this article. The performance of the hybrid method suggests that competition agents should also be *risk sensitive*, as the algorithm that accrues the most cumulative reward in expectation may not be preferable. When performance varies greatly from run to run, the chance of winning the competition can be higher using a *risk-averse* agent with lower expected performance but also lower variance. Existing methods for risk-sensitive reinforcement learning [24, 53], are not applicable to policy-search approaches like neuroevolution and require as input a quantification of the risk sensitivity. In competition settings, the optimal amount of risk aversion is a function of known factors such as the number of test runs so it may be possible to develop methods that automatically determine their own risk sensitivity.

In addition, though neuroevolution proved ineffective in improving on the manually designed network topology, more sophisticated representation-learning approaches may fare better. Recent advances in *indirect encodings* such as HyperNEAT [22, 23, 75] could further improve performance. Though the selection races approach used here proved effective, additional efficiencies may be obtainable using new methods for selecting multiple arms in a k-armed bandit problem [42]. In addition, these approaches would benefit from mechanisms that automatically select an appropriate target confidence level, by weighing the cost of additional evaluations against the expected long-term benefit to the course of evolution.

Finally, the generalized helicopter hovering problem itself could be further extended. While FGHH remedies many of the shortcomings of GHH-08 and GHH-09, it still contains simplifying assumptions. For example, though the transition dynamics are noisy, the agent's observations are not. In some settings, noise in sensors could lead to substantial partial observability even when using sophisticated motion-capture systems, yielding a more difficult control problem. Also, while obtaining flight data in FGHH is risky, it can be completed in a single episode. A more challenging problem would require the agent to constantly refine its model as new flight data arrives and thus integrate exploration, learning, and acting throughout each run.

8 Conclusion

We presented an extended case study in the application of neuroevolution to generalized simulated helicopter hovering, considering three increasingly challenging variations of the task. The results demonstrate that (1) neuroevolution can be effective for such complex on-line reinforcement learning tasks, (2) neuroevolution excels at finding effective policies but not at learning helicopter models, (3) due to the difficulty of learning reliable helicopter models, model-based approaches to helicopter hovering are feasible only when domain expertise is available to aid the design of a suitable model representation, (4) model-based approaches are superior only if the domain expertise needed to design a suitable model representation is present and (5) recent advances in efficient resampling can enable neuroevolution to tackle more aggressively generalized reinforcement learning tasks. The results also illustrate the importance of on-line learning for evolutionary approaches to reinforcement learning, how expensive exploration can be in realistic problems, and the challenges of designing suitable generalized tasks. Finally, this research points the way to future work in risk-sensitive reinforcement learning, efficient resampling, and still more challenging benchmarks in generalized helicopter control.

Table 6 Neuroevolution parameter settings for GHH-08 and GHH-09

`pop_size`	50	`mutate_prob`	0.75
`elite_size`	49	`mutate_frac`	0.1
`plateau_threshold`	1,000	`mutate_std`	0.8
`crossover_prob`	0.5	`mutate_repl`	0.25
`averaging_prob`	0.5		

Appendix A: Neuroevolution for GHH-08 and GHH-09

To evolve policies, we employ a simple steady state neuroevolutionary method. Each neural network is encoded as a genome consisting of a vector of the network's weights. An initial population of size `pop_size` is formed by repeatedly applying weight mutations to a given prototype network. The fraction of weights being mutated is given by `mutate_frac`, i.e., it specifies the probability that an individual weight is altered. When weight mutations occur, their magnitude is sampled from a Gaussian distribution with mean 0.0 and standard deviation `mutate_std`. With probability `mutate_repl`, 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.

Next, the `elite_size` best performing networks are copied to the next generation and the rest are replaced with new individuals. With probability `crossover_prob`, each new offspring 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.

Table 6 lists the parameter settings of this algorithm used in our experiments in GHH-08 and GHH-09. 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.

Table 7 Neuroevolution parameter settings for FGHH

pop_size	50	mutate_prob	0.75
select_size	20	mutate_frac	0.1
generations	4,000	mutate_std	0.8
crossover_prob	1.0	mutate_repl	0.25
averaging_prob	0.5		

Appendix B: Neuroevolution for FGHH

When evolving generic policies for FGHH, we employ a slightly altered neuroevolutionary algorithm. The main difference is that the algorithm is generational, instead of steady state. This change facilitates the use of selection races, which require a generational approach. At the end of each generation, the `select_size` policies with the highest estimated fitness are selected for reproduction. Note that `pop_size` and `select_size` correspond to λ and μ, respectively, in Algorithm 1. In addition, instead of setting a plateau threshold for termination, evolution runs for a fixed number of `generations`. Finally, all individuals are produced via crossover (`crossover_prob` = 1). Table 7 lists the parameter settings of this algorithm used in all our experiments in FGHH.

References

1. Abbeel P, Coates A, Ng A (2010) Autonomous helicopter aerobatics through apprenticeship learning. Int J Robotics Res 29(13):1608–1639
2. Abbeel P, Coates A, Quigley M, Ng AY (2007) An application of reinforcement learning to aerobatic helicopter flight. In: Advances in neural information processing systems 19. MIT Press, Cambridge, pp 1–8
3. Abbeel P, Ganapathi V, Ng AY (2006) Learning vehicular dynamics with application to modeling helicopters. In: Proceedings of neural information processing systems (NIPS)
4. Abbeel P, Ng A (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on machine learning
5. Abbeel P, Ng AY (2005) Exploration and apprenticeship learning in reinforcement learning. In: Proceedings of the twenty-first international conference on machine learning
6. Bagnell J, Schneider J (2001) Autonomous helicopter control using reinforcement learning policy search methods. In: Proceedings of the IEEE international conference on robotics and automation 2001
7. Beielstein T, Markon S (2002) Threshold selection, hypothesis tests and DOE methods. In: 2002 congress on evolutionary computation, pp 777–782
8. Bellman RE (1957) Dynamic programming. Princeton University Press, Princeton
9. Bellman RE (1957) A Markov decision process. J Math Mech 6:679–684
10. Brafman R, Tennenholtz M, Schuurmans D (2003) R-max-A general polynomial time algorithm for near-optimal reinforcement learning. J Mach Learn Res 3(2):213–231
11. Branke J, Schmidt C (2003) Selection in the presence of noise. In: Proceedings of the genetic and evolutionary computation conference (GECCO), pp 766–777
12. Branke J, Schmidt C (2004) Sequential sampling in noisy environments. In: Proceedings of the international conference on parallel problem solving from nature (PPSN), pp 202–211
13. Butz M, Goldberg D, Lanzi P (2005) Gradient descent methods in learning classifier systems: improving XCS performance in multistep problems. IEEE Trans Evolut Comput 9(5)
14. Cardamone L, Loiacono D, Lanzi P (2009) On-line neuroevolution applied to the open racing car simulator. In: Proceedings of the congress on evolutionary computation (CEC), pp 2622–2629
15. Cardamone L, Loiacono D, Lanzi PL (2010) Learning to drive in the open racing car simulator using online neuroevolution. Comput Intell AI in Games IEEE Trans 2(3):176–190
16. Chen S, Wu Y, Luk B (2002) Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks. Neural Netw IEEE Trans 10(5):1239–1243
17. De Boer P, Kroese D, Mannor S, Rubinstein R (2005) A tutorial on the cross-entropy method. Ann Oper Res 134(1):19–67
18. De Nardi R, Holland O (2006) Ultraswarm: a further step towards a flock of miniature helicopters. In: Proceedings of the SAB workshop on swarm robotics. Springer, Berlin, pp 116–128
19. De Nardi R, Holland O (2008) Coevolutionary modelling of a miniature rotorcraft. In: IAS-10: intelligent autonomous systems conference, p 364
20. Floreano D, Mondada F (2002) Evolution of homing navigation in a real mobile robot. IEEE Trans Syst Man Cybern B 26(3):396–407
21. Floreano D, Urzelai J (2001) Evolution of plastic control networks. Auton Robots 11(3):311–317
22. Gauci J, Stanley KO (2008) A case study on the critical role of geometric regularity in machine learning. In: Proceedings of the twenty-third AAAI conference on artificial intelligence
23. Gauci J, Stanley KO (2010) Autonomous evolution of topographic regularities in artificial neural networks. Neural Comput 22(7):1860–1898
24. Geibel P, Wysotzki F (2005) Risk-sensitive reinforcement learning applied to control under constraints. J Artif Intell Res 24(1):81–108
25. Goldberg DE, Deb K, Clark JH (1991) Genetic algorithms, noise, and the sizing of populations. Complex Syst 6:333–362
26. Goldberg D, Rudnick M (1991) Genetic algorithms and the variance of fitness. Complex Syst 5(3):265–278
27. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning, 1st edn. Addison-Wesley, Boston
28. Gomez F, Schmidhuber J, Miikkulainen R (2006) Efficient nonlinear control through neuroevolution. In: Proceedings of the European conference on machine learning
29. Gruau F, Whitley D, Pyeatt L (1996) A comparison between cellular encoding and direct encoding for genetic neural networks. In: Genetic programming 1996: Proceedings of the first annual conference, pp 81–89
30. Hansen N, Müller S, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolut Comput 11(1):1–18
31. Harik G, Cantú-Paz E, Goldberg D, Miller B (1999) The gambler's ruin problem, genetic algorithms, and the sizing of populations. Evolut Comput 7(3):231–253
32. Heidrich-Meisner V, Igel C (2009) Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In: Proceedings of the 26th annual international conference on machinelearning, ACM, pp 401–408

33. Hernandez-Diaz A, Coello C, Perez F, Caballero R, Molina J, Santana-Quintero L (2008) Seeding the initial population of a multi-objective evolutionary algorithm using gradient-based information. In: evolutionary computation, 2008. CEC 2008. (IEEE world congress on computational intelligence). IEEE congress on, pp 1617–1624
34. Hoffmann G, Huang H, Waslander S, Tomlin C (2007) Quadrotor helicopter flight dynamics and control: theory and experiment. In: Proceedings of the AIAA guidance, navigation, and control conference, pp 1–20
35. Hurst J, Bull L (2006) A neural learning classifier system with self-adaptive constructivism for mobile robot control. Artif Life 12(3):353–380
36. Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput Fusion Found Methodol Appl 9(1):3–12
37. Jin Y, Olhofer M, Sendhoff B (2002) A framework for evolutionary optimization with approximate fitness functions. IEEE Trans Evolut Comput 6(5):481–494
38. Julstrom BA (1994) Seeding the population: improved performance in a genetic algorithm for the rectilinear steiner problem. In: Proceedings of the 1994 ACM symposium on applied computing, SAC ’94, pp 222–226
39. Kaelbling LP (1993) Learning in embedded systems. MIT Press, Cambridge
40. Kaelbling LP, Littman ML, Moore AP (1996) Reinforcement learning: A survey. J Art Intell Res 4:237–285
41. Kalyanakrishnan S, Stone P (2009) An empirical analysis of value function-based and policy search reinforcement learning. In: Proceedings of the eighth international joint conference on autonomous agents and multi–agent systems (AAMAS 2009)
42. Kalyanakrishnan S, Stone P (2010) Efficient selection of multiple bandit arms: theory and practice. In: Proceedings of the twenty-seventh international conference on machine learning (ICML 2010) (to appear)
43. Kassahun Y, Sommer G (2005) Efficient reinforcement learning through evolutionary acquisition of neural topologies. In: 13th European symposium on artificial neural networks, Bruges, Belgium, pp 259–266
44. Kearns M, Singh S (1998) Near-optimal reinforcement learning in polynomial time. In: Proceedings of the 15th international conference on machine learning. Morgan Kaufmann, San Francisco, pp 260–268
45. Kernbach S, Meister E, Scholz O, Humza R, Liedke J, Ricotti L, Jemai J, Havlik J, Liu W (2009) Evolutionary robotics: the next-generation-platform for on-line and on-board artificial evolution. In: CEC’09: IEEE congress on evolutionary computation, pp 1079–1086
46. Koppejan R (2009) Neuroevolutionary reinforcement learning for generalized helicopter control. Master’s thesis, Universiteit van Amsterdam
47. Koppejan R, Whiteson S (2009) Neuroevolutionary reinforcement learning for generalized helicopter control. In: GECCO 2009: Proceedings of the genetic and evolutionary computation conference, pp 145–152
48. Lanzi PL, Colombetti M (1999) An extension to the XCS classifier system for stochastic environments. In: GECCO-99: Proceedings of the genetic and evolutionary computation conference, pp 353–360
49. Lupashin S, Schollig A, Sherback M, D’Andrea R (2010) A simple learning strategy for high-speed quadrocopter multi-flips. In: ICRA-10: IEEE international conference on robotics and automation, pp 1642–1648
50. Maron O, Moore AW (1997) The racing algorithm: model selection for lazy learners. Artificial Intelligence Review 11(1–5):193–225
51. Martín HJA, de Lope J (2009) Learning autonomous helicopter flight with evolutionary reinforcement learning. In: 12th international conference on computer aided systems theory (EUROCAST), pp 75–82
52. Meyer J, Husbands P, Harvey I (1998) Evolutionary robotics: a survey of applications and problems. In: evolutionary robotics. Springer, pp 1–21
53. Mihatsch O, Neuneier R (2002) Risk-sensitive reinforcement learning. Mach Learn 49(2):267–290
54. Moore A, Atkeson C (1993) Prioritized sweeping: reinforcement learning with less data and less real time. Mach Learn 13:103–130
55. Moriarty DE, Schultz AC, Grefenstette JJ (1999) Evolutionary algorithms for reinforcement learning. J Art Intell Res 11:199–229
56. Ng A, Jordan M (2000) PEGASUS: a policy search method for large MDPs and POMDPs. In: Proceedings of the sixteenth conference on uncertainty in artificial intelligence, pp 406–415
57. Ng A.Y, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, Berger E, Liang E (2004) Inverted autonomous helicopter flight via reinforcement learning. In: Proceedings of the international symposium on experimental robotics
58. Nordin P, Banzhaf W (1997) An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. Adapt Behav 5(2):107
59. Ong Y, Nair P, Keane A (2003) Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA J 41(4):687–696
60. Oyekan J, Lu B, Li B, Gu D, Hu H (2010) A behavior based control system for surveillance UAVs. In: Liu H, Gu D, Howlett RJJ, Liu Y (eds) Robot intelligence, advanced information and knowledge processing. Springer, Berlin, pp 209–228
61. Poli R, Cagnoni S (1997) Genetic programming with user-driven selection: experiments on the evolution of algorithms for image enhancement. In: Proceedings of the second annual conference on genetic programming, pp 269–277
62. Ponterosso P, Fox DSJ (1999) Heuristically seeded genetic algorithms applied to truss optimisation. Eng Comput 15:345–355
63. Poupart P, Vlassis N, Hoey J, Regan K (2006) An analytic solution to discrete Bayesian reinforcement learning. In: Proceedings of the twenty-third international conference on machine learning
64. Pratihar D (2003) Evolutionary robotics: a review. Sadhana 28(6):999–1009
65. Priesterjahn S, Weimer A, Eberling M (2008) Real-time imitation-based adaptation of gaming behaviour in modern computer games. In: Proceedings of the genetic and evolutionary computation conference, pp 1431–1432
66. Purwin O, D’Andrea R (2009) Performing aggressive maneuvers using iterative learning control. In: ICRA-09: IEEE international conference on robotics and automation, 2009, pp 1731–1736
67. Regis R, Shoemaker C (2004) Local function approximation in evolutionary algorithms for the optimization of costly functions. IEEE Trans Evolut Comput 8(5):490–505
68. Sastry K, Lima CF, Goldberg DE (2006) Evaluation relaxation using substructural information and linear estimation. In: Proceedings of the 8th annual conference on genetic and evolutionary computation, GECCO ’06, pp 419–426
69. Schmidt M, Lipson H (2006) Actively probing and modeling users in interactive coevolution. In: Proceedings of the 8th conference on genetic and evolutionary computation, pp 385–386
70. Schmidt M, Lipson H (2008) Coevolution of fitness predictors. IEEE Trans Evolut Comput 12(6):736–749
71. Schroder P, Green B, Grum N, Fleming P (2001) On-line evolution of robust control systems: an industrial active magnetic bearing application. Cont Eng Pract 9(1):37–49

72. Siebel NT, Sommer G (2007) Evolutionary reinforcement learning of artificial neural networks. Int J Hybrid Intell Syst 4(3):171–183
73. Sigaud O, Wilson S (2007) Learning classifier systems: a survey. Soft Comput Fusion Found Method Appl 11(11):1065–1078
74. Stagge P (1998) Averaging efficiently in the presence of noise. Parallel Probl Solving Nat 5:188–197
75. Stanley KO, D'Ambrosio DB, Gauci J (2009) A hypercube-based indirect encoding for evolving large-scale neural networks. Art Life 15(2):185–212
76. Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolut Comput 10(2):99–127
77. Steels L (1994) Emergent functionality in robotic agents through on-line evolution. In: artificial life IV: Proceedings of the fourth international workshop on the synthesis and simulation of living systems, pp 8–16
78. Stone P, Sutton RS, Kuhlmann G (2005) Reinforcement learning in Robocup-soccer keepaway. Adapt Behav 13(3):165–188
79. Strehl AL, Li L, Wiewiora E, Langford J, Littman ML (2006) PAC model-free reinforcement learning. In: ICML-06: Proceedings of the 23rd international conference on machine learning, pp 881–888
80. Sutton RS (1988) Learning to predict by the methods of temporal differences. Mach Learn 3:9–44
81. Sutton RS (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings of the seventh international conference on machine learning, pp 216–224
82. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
83. Tan C, Ang J, Tan K, Tay A (2008) Online adaptive controller for simulated car racing. In: Congress on evolutionary computation (CEC), pp 2239–2245
84. Tang J, Singh A, Goehausen N, Abbeel P (2010) Parameterized maneuver learning for autonomous helicopter flight. In: International conference on robotics and automation (ICRA)
85. Tanner B, White A (2009) RL-Glue : Language-independent software for reinforcement-learning experiments. J Mach Learn Res 10:2133–2136
86. Tesauro G (1995) Temporal difference learning and TD-gammon. Commun ACM 38(3):58–68. doi:10.1145/203330.203343
87. Watkins C (1989) Learning from delayed rewards. Ph.D. thesis, Cambridge University
88. Whiteson S, Kohl N, Miikkulainen R, Stone P (2005) Evolving keepaway soccer players through task decomposition. Mach Learn 59(1):5–30
89. Whiteson S, Stone P (2006) Evolutionary function approximation for reinforcement learning. J Mach Learn Res 7:877–917
90. Whiteson S, Stone P (2006) On-line evolutionary computation for reinforcement learning in stochastic domains. In: GECCO 2006: Proceedings of the genetic and evolutionary computation conference, pp 1577–1584
91. Whiteson S, Tanner B, Taylor ME, Stone P (2009) Generalized domains for empirical evaluations in reinforcement learning. In: ICML 2009: Proceedings of the twenty-sixth international conference on machine learning: workshop on evaluation methods for machine learning
92. Whiteson S, Tanner B, Taylor ME, Stone P (2011) Protecting against evaluation overfitting in empirical reinforcement learning. In: ADPRL 2011: Proceedings of the IEEE symposium on adaptive dynamic programming and reinforcement learning (to appear)
93. Whiteson S, Tanner B, White A (2010) The reinforcement learning competitions. AI Mag 31(2):81–94
94. Whiteson S, Taylor ME, Stone P (2010) Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning. Auton Agents Multi-Agent Syst 21(1):1–27
95. Wilson A, Fern A, Ray S, Tadepalli P (2007) Multi-task reinforcement learning: a hierarchical Bayesian approach. In: Proceedings of the 24th international conference on machine learning, pp 1015–1022
96. Wilson S (1995) Classifier fitness based on accuracy. Evolut Comput 3(2):149–175
97. Wilson S (2001) Function approximation with a classifier system. In: GECCO-2001: Proceedings of the genetic and evolutionary computation conference, p 974
98. Yang D, Flockton S (1995) Evolutionary algorithms with a coarse-to-fine function smoothing. In: IEEE international conference on evolutionary computation 2: 657–662
99. Yao X (1999) Evolving artificial neural networks. Proc IEEE 87(9):1423–1447
100. Zufferey J-C, Floreano D, Van Leeuwen M, Merenda T (2002) Evolving vision-based flying robots. In: Lee B, Wallraven P (eds) Proceedings of the 2nd international workshop on biologically motivated computer vision (BMCV). Springer, Berlin, pp 592–600