To appear in *Adaptive Behavior*, 15(1), 2007.

Empirical Studies in Action Selection with Reinforcement Learning

Shimon Whiteson
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-1188
shimon@cs.utexas.edu
http://www.cs.utexas.edu/~shimon

Matthew E. Taylor
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-1188
mtaylor@cs.utexas.edu
http://www.cs.utexas.edu/~mtaylor

Peter Stone
Department of Computer Sciences
The University of Texas at Austin
1 University Station C0500
Austin, TX 78712-1188
pstone@cs.utexas.edu
http://www.cs.utexas.edu/~pstone

October 4, 2006

Abstract

To excel in challenging tasks, intelligent agents need sophisticated mechanisms for action selection: they need policies that dictate what action to take in each situation. Reinforcement learning (RL) algorithms are designed to learn such policies given only positive and negative rewards. Two contrasting approaches to RL that are currently in popular use are temporal difference (TD) methods, which learn value functions, and evolutionary methods, which optimize populations of candidate policies. Both approaches have had practical successes but few studies have directly compared them. Hence, there are no general guidelines describing their relative strengths and weaknesses. In addition, there has been little cross-collaboration, with few attempts to make them work together or to apply ideas from one to the other. This article aims to address these shortcomings via three empirical studies that compare these methods and investigate new ways of making them work together.

First, we compare the two approaches in a benchmark task and identify variations of the task that isolate factors critical to each method's performance. Second, we investigate ways to make evolutionary algorithms excel at on-line tasks by borrowing exploratory mechanisms traditionally used by TD methods. We present empirical results demonstrating a dramatic performance improvement. Third, we explore a novel way of making evolutionary and TD methods work together by using evolution to automatically discover good representations for TD function approximators. We present results demonstrating that this novel approach can outperform both TD and evolutionary methods alone.

Keywords: reinforcement learning, temporal difference methods, evolutionary computation, neural networks, robot soccer, autonomic computing

1 Introduction

A primary focus of artificial intelligence is the design of intelligent agents. To succeed, such agents must tackle two main challenges, as described by Prescott, Bryson, and Seth (2007): 1) identifying the most useful set of available actions, called *action specification*, and 2) finding a good *policy* that indicates what action to take in each situation, called *action selection*. This article focuses on the latter problem by investigating how such policies can be discovered via *reinforcement learning* (RL) (Sutton & Barto, 1998). In RL, agents never see examples of correct or incorrect behavior but instead must learn policies using a much weaker form of feedback: positive and negative delayed rewards. By interacting with their environment, such agents strive to learn policies that maximize the reward they accrue over the long term.

Two contrasting approaches to RL that are currently in popular use are *temporal difference* (TD) methods (Sutton, 1988) and *evolutionary algorithms* (Moriarty, Schultz, & Grefenstette, 1999). Temporal difference methods rely on the concept of *value functions*, which indicate, for a particular policy, the long-term value of taking a particular action in a particular situation, or *state*. By combining principles of dynamic programming with statistical sampling, TD methods use the immediate rewards received by an agent to incrementally improve its estimated value function and thereby its policy. By contrast, evolutionary algorithms for RL do not reason about value functions. Instead, they simulate the process of natural selection to optimize a population of candidate policies. Like other policy search methods (Sutton, McAllester, Singh, & Mansour, 2000a; Ng & Jordan, 2000; Kohl & Stone, 2004), they directly search the space of policies for those that achieve the best performance.

Although these two approaches to learning action selection policies have both had success in difficult RL tasks, only a few studies (e.g. Runarsson & Lucas, 2005; Gomez & Miikkulainen, 2002; Moriarty & Miikkulainen, 1996; Pollack, Blair, & Land, 1997; Whitley, Dominic, Das, & Anderson, 1993) have directly compared them. While these comparisons present useful data, they do not isolate the factors critical to the performance of each method. As a result, there are currently no general guidelines describing the methods' relative strengths and weaknesses. In addition, since the two research communities are largely disjoint and often focus on different problems, there has been little cross-collaboration, with few attempts to make these different methods work together or to apply ideas from one to the other. In this article, we present research that aims to address these shortcomings by conducting rigorous empirical comparisons between TD and evolutionary methods and investigating new methods for making them work together. In particular, we present three empirical studies that compare, contrast, and even combine these two methods.

First, we apply NEAT (Stanley & Miikkulainen, 2002), one of the most successful evolutionary RL methods, to *3 vs. 2 Keepaway* (Stone, Kuhlmann, Taylor, & Liu, 2006), a robot soccer benchmark task at which TD methods have previously excelled (Kostiadis & Hu, 2001; Stone, Sutton, & Kuhlmann, 2005). Our results in this domain demonstrate that NEAT discovers better policies than Sarsa (Rummery & Niranjan, 1994; Singh & Sutton, 1996), the best performing TD method, though it requires many more evaluations to do so. We also compare NEAT and Sarsa in two variations of Keepaway designed to isolate factors critical to the performance of each method and find that they have contrasting strengths and weaknesses. Together, these results shed light on the open question of when evolutionary or TD methods perform better and why.

Second, we investigate ways to make evolutionary algorithms excel at *on-line* reinforcement leaning tasks, i.e. those in which the agent learns, not in a safe environment like a simulator, but in the real world. TD methods naturally excel at on-line tasks because they have explicit mechanisms for balancing *exploration* and *exploitation*. Evolutionary algorithms do not. We investigate a novel way to borrow these mechanisms and apply them to evolutionary methods. We present experiments in the benchmark mountain car domain (Sutton & Barto, 1998) and server job scheduling (Whiteson & Stone, 2006a), a challenging stochastic domain drawn from the field of *autonomic computing* (Kephart & Chess, 2003). The results demonstrate that this approach can dramatically improve the on-line performance of evolutionary methods for RL.

Third, we explore a novel way of making evolutionary and TD methods work together called *evolutionary function approximation*. While evolutionary methods are typically used to learn policies directly, we use them to automatically discover good representations for TD *function approximators*, which describe value functions using concise, parameterized functions. Hence, this novel synthesis *evolves* agents that are better able to *learn*. In particular, we present NEAT+Q, which uses NEAT to evolve neural network function approximators for Q-learning (Watkins, 1989), a popular TD method. Additional experiments in the mountain car and server job scheduling domains demonstrate that NEAT+Q can outperform both regular NEAT and Q-learning with manually designed representations.

Together, these empirical studies contribute to our understanding of action selection in RL by 1) providing some much-needed empirical comparisons between evolutionary and TD methods and 2) examining novel combinations of these two approaches. The results demonstrate that good action selection strategies can be found, not only by choosing between evolutionary and TD methods, but by making them work together.

2 Background

In this section, we present brief overviews of the TD and evolutionary methods used in this article.

2.1 Temporal Difference Methods

TD methods rely on value functions to perform action selection. For a given policy, a value function $Q(s, a)$ estimates the long-term value of taking an action a in a state s. Each time the agent takes an action, TD methods incrementally refine both the agent's policy and the estimated value function for that policy. In the simple case, the value function is stored in a table, with one entry for each state-action pair.

In this article, we use Sarsa (Rummery & Niranjan, 1994; Singh & Sutton, 1996) and Q-learning (Watkins, 1989) as representative TD methods. We choose them because they are well-established, canonical methods with numerous empirical successes (Sutton, 1996; Crites & Barto, 1998; Stone et al., 2005). The update rule for tabular Sarsa is:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma Q(s', a')) \tag{1}$$

where s' and a' are the agent's next state and action, respectively, α is the learning rate and γ is a discount factor used to weight immediate rewards more heavily than future rewards. Q-learning is a simple variation of Sarsa in which the update is based, not on the action the agent took in the subsequent state, but on the *optimal* action available in that state. Hence, its update rule is:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a')) \tag{2}$$

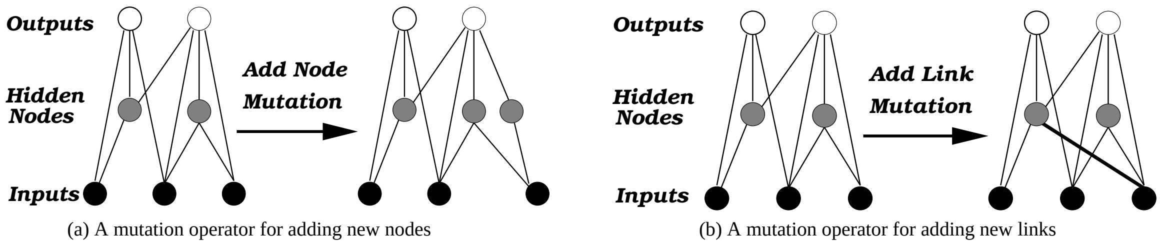


Figure 1: Examples of NEAT's structural mutation operators. In (a), a hidden node is added by splitting a link in two. In (b), a link, shown with a thicker black line, is added to connect two nodes.

Since Q-learning's update rule is independent of the policy the agent is following, it is an *off-policy* TD method, a feature which facilitates proofs of convergence. In practice, both Sarsa and Q-learning are known to perform well.

In domains or those with large or continuous state space, the value function cannot be represented in a table. In such cases, TD methods rely on function approximators, which represent the mapping from state-action pairs to values via more concise, parameterized functions. Many function approximators are currently in use; in this article we use radial basis functions (RBFs) (Powell, 1987) and neural networks (Crites & Barto, 1998).

2.2 NeuroEvolution of Augmenting Topologies (NEAT)[1]

Unlike TD methods, evolutionary approaches to RL do not learn value functions. Instead, they simulate the process of natural selection to optimize a population of candidate policies. Like other policy search methods (Sutton et al., 2000a; Ng & Jordan, 2000; Kohl & Stone, 2004), they directly search the space of policies for those that achieve the best performance. In this article, we use NeuroEvolution of Augmenting Topologies (NEAT) as a representative evolutionary method. NEAT is an appropriate choice because of its empirical successes on difficult RL tasks like pole balancing (Stanley & Miikkulainen, 2002) and robot control (Stanley & Miikkulainen, 2004a). In addition, NEAT is appealing because, unlike many other optimization techniques, it automatically learns an appropriate representation for the solution.

In a typical neuroevolutionary system (Yao, 1999), the weights of a neural network are strung together to form an individual genome. A population of such genomes is then evolved by evaluating each one and selectively reproducing the fittest individuals through crossover and mutation. Most neuroevolutionary systems require the designer to manually determine the network's topology (i.e. how many hidden nodes there are and how they are connected). By contrast, NEAT automatically evolves the topology to fit the complexity of the problem. It combines the usual search for network weights with evolution of the network structure.

Unlike other systems that evolve network topologies and weights, NEAT begins with a uniform population of simple networks with no hidden nodes and inputs connected directly to outputs. In addition to normal weight mutations, NEAT also has *structural mutations*, shown in Figure 1, which add hidden nodes and links to the network. Only those structural mutations that improve performance tend to survive; in this way, NEAT searches through a minimal number of weight dimensions and finds the appropriate level of complexity for the problem.

Since NEAT is a general purpose optimization technique, it can be applied to a wide variety of problems. For reinforcement learning tasks, NEAT typically evolves *action selectors*, which directly

[1]Section 2.2 is adapted from the original NEAT paper (Stanley & Miikkulainen, 2002).

map states to the action the agent should take in that state. Hence, policies are represented directly, not via value functions as in TD methods.

3 Comparing NEAT and Sarsa Approaches to Action Selection

Although both evolutionary and TD methods have had success in difficult RL tasks, only a few studies (e.g. Runarsson & Lucas, 2005; Gomez & Miikkulainen, 2002; Moriarty & Miikkulainen, 1996; Pollack et al., 1997; Whitley et al., 1993) have directly compared them. While these comparisons present useful data, they do not isolate the factors critical to the performance of each method. As a result, there are currently no general guidelines describing the methods' relative strengths and weaknesses. The goal of our first empirical study is to address this shortcoming by performing a rigorous comparison of the two approaches, not only on a benchmark RL task, but on variations of that task designed to reveal which factors have the greatest effect on performance.

No such comparison can ever be conclusive because the results depend on the specific domains tested. Nonetheless, careful experiments can contribute substantially to our understanding of which methods to use when. In our first empirical study, we conduct experiments in *3 vs. 2 Keepaway* (Stone et al., 2006), a standard RL benchmark domain based on robot soccer. Keepaway is an appealing platform for empirical comparisons because the performance of TD methods in it has already been established in previous studies (Kostiadis & Hu, 2001; Stone et al., 2005). While evolutionary methods have been applied to variations of Keepaway (Hsu & Gustafson, 2002; Whiteson, Kohl, Miikkulainen, & Stone, 2005), they have never been applied to the benchmark version of the task.

Empirical comparisons are limited not just by the domain but also by the specific methods tested. Many evolutionary and TD methods are currently in use and it is not feasible to test them all. In this empirical study, we evaluate Sarsa with RBF function approximators, the best performing TD method to date in this domain (Stone et al., 2006), and compare its performance to NEAT, which has a strong record on challenging RL tasks (Stanley & Miikkulainen, 2002, 2004a, 2004b). We present results, not just in the benchmark version of Keepaway, but in two variations designed to isolate factors critical to the performance of each method.

3.1 The Benchmark Keepaway Task

Keepaway is a subproblem of the full 11 vs. 11 simulated soccer game in which a team of three *keepers* attempts to maintain possession of the ball while two *takers* attempt to steal the ball or force it out of bounds, ending an *episode*. Figure 2 depicts three keepers playing against two takers. The agents choose not from the simulator's primitive actions but from a set of higher-level macro-actions implemented as part of the player. These macro-actions can last more than one time step and the keepers make decisions only when a macro-action terminates. The macro-actions are Hold Ball, Get Open, Receive, and Pass (Stone et al., 2005). The agents make decisions at discrete time steps, at which point macro-actions are initiated and terminated. Takers do not learn and always follow a static hand-coded strategy.

The keepers learn in a constrained policy space: they have the freedom to select actions only when in possession of the ball. A keeper in possession may either hold the ball or pass to one of its teammates. Therefore in 3 vs. 2 Keepaway, a keeper with the ball may select among 3 actions: Hold Ball, Pass to Closest Teammate, or Pass to Farthest Teammate. Keepers not in possession of the ball are required to execute the Receive macro-action in which the keeper who can reach the ball the fastest goes to the ball and the remaining players use the Get Open macro-action.

To solve this task using NEAT, we trained a population of 100 neural network action selectors, each with 13 inputs, one for each state feature, and 3 outputs, one for each macro-action. When a given network controls the keepers, their actions correspond to the output with the highest activation. Its performance in a given episode is its *hold time*: the number of seconds the keepers control the ball before it goes out of bounds or is stolen by a taker. Since Keepaway is a stochastic domain (with noisy sensors and actuators), each network was evaluated for 60 episodes on average to obtain accurate fitness estimates. In informal experiments, fewer episodes per network caused evolution to stagnate while more episodes led to slower learning. NEAT parameters were set the same as reported in previous work (Taylor, Whiteson, & Stone, 2006).

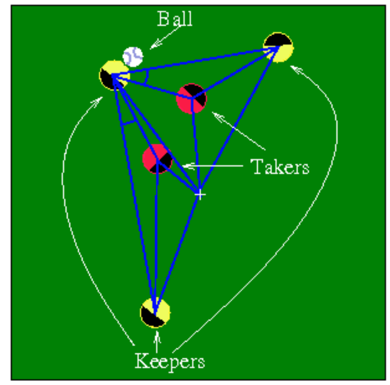


Figure 2: The 13 state variables used in Keepaway, including 11 distances between players and the center of the field and 2 angles along passing lanes.

To solve this task using Sarsa, we used RBF function approximators with Gaussian basis functions of width $\sigma = 0.25$. While other types of function approximators, including CMACs and neural networks, have been successfully applied to this task, we use RBFs because they have achieved the best performance to date (Stone et al., 2006). A learning rate of $\alpha = 0.05$, an ϵ-greedy exploration strategy with $\epsilon = 0.01$ were likewise used because previous studies found them to be effective in the Keepaway domain.

The RoboCup Soccer Server's time steps are in 0.1 second increments and all times reported below refer to simulator time. Thus we report only sample complexity and not computational complexity; the running time for our learning methods is negligible compared to that of the Soccer Server. The machines used for our experiments allowed us to speed up the simulator by a factor of two so that the real experimental time required was roughly half the reported simulator time.

Figure 3 shows the results of our experiments comparing these two methods. The graph plots, at regular intervals, the hold time of the best policy discovered so far, averaged over 1,000 episodes. These results are also averaged over 20 independent trials for Sarsa and 5 for NEAT. Each trial was run until performance plateaued. Since the Sarsa runs plateau much sooner than the NEAT curves, we were able to conduct more of them. Note that the learning curve is extended on the graph even after learning has finished, denoted by a horizontal performance line without plotted data points.

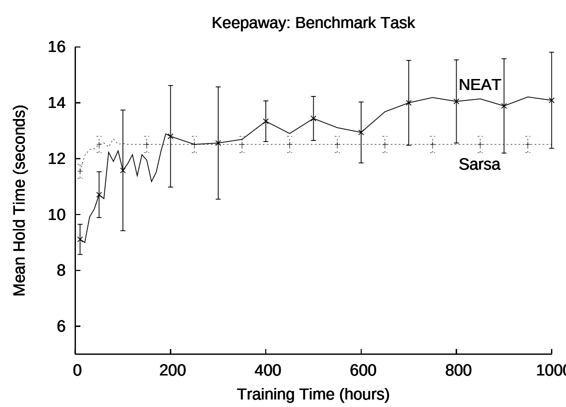


Figure 3: Mean hold times of the best policies learned by NEAT and Sarsa in the benchmark Keepaway task. Error bars show 95% confidence intervals.

The results demonstrate that NEAT can learn better policies in the benchmark Keepaway task than Sarsa with RBFs. Unpaired Student's t-tests conducted at regular intervals confirm that, after 650 hours of training, the difference in average hold times is statistically significant with 95% confidence, despite slightly overlapping confidence intervals. These results also highlight an important trade-off between the two methods. While NEAT ultimately learns better policies, it requires many more evaluations to do so. Sarsa learns much more rapidly and in the early part of learning its average policy is much better than NEAT's.

In order to discover what characteristics of the Keepaway task most critically affect the speed and quality of learning in each method, we conducted additional experiments in two variations of the Keepaway task. The remainder of this section describes these variations and the results of our experiments therein.

3.2 The Fully Observable Keepaway Task

In the benchmark Keepaway task, the agents' sensors are noisy. Since the agents can only partially observe the true state of the world, the task is non-Markovian, i.e. the probability distribution over next states is not independent of the agents' state and action histories. This fact could be problematic for Sarsa since TD update rules assume the environment is Markovian, though in practice they can still perform well when it is not (Sutton & Barto, 1998). By contrast, NEAT can evolve recurrent networks that cope with non-Markovian tasks by recording important information about previous states (Stanley & Miikkulainen, 2002; Gomez & Schmidhuber, 2005). Therefore, we hypothesized that Sarsa's relative performance would improve if sensor noise was removed, rendering the Keepaway task fully observable and effectively Markovian.[2]

To test this hypothesis we conducted 20 trials of Sarsa and 5 trials of NEAT in the fully observable Keepaway task. Figure 4 shows the results of these experiments. As in the benchmark version of the task, Sarsa learns much more rapidly than NEAT. However, in the fully observable version, Sarsa also learns substantially better policies. Unpaired Student's t-tests confirm that the difference in average hold times is statistically significant with 95% confidence for all points graphed.

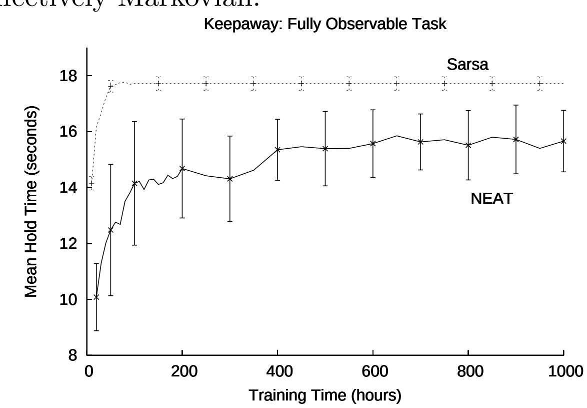


Figure 4: Mean hold times of the best policies learned by NEAT and Sarsa in the fully observable Keepaway task. Error bars show 95% confidence intervals.

These results verify our hypothesis that full observability is a critical factor in Sarsa's performance in the Keepaway task. While Sarsa can learn well in the partially observable benchmark version, its performance relative to NEAT improves dramatically when sensor noise is removed. These results are not surprising given the way these two methods work: one of Sarsa's underlying assumptions is violated in the absence of the Markov property. By contrast, NEAT makes no such assumption and therefore tasks with partial observability are not particularly problematic.

[2]The state is still not truly Markovian because player velocities are not included. However, the Keepaway benchmark task does not include velocity because past research found that it was not useful for learning, since players have low inertia and the field has a high coefficient of friction. Hence, we use the same state variables as previous work (Stone et al., 2005, 2006) but note that without sensor noise the state is "effectively Markovian."

3.3 The Deterministic Keepaway Task

Even in the fully observable version of the task, which has no sensor noise, the Keepaway task remains highly stochastic due to noise in the agents' actuators and randomness in the agents' initial states. In both the benchmark task and the fully observable variation described above, this stochasticity greatly slows NEAT's learning rate. A policy's measured fitness in a given episode has substantial variance and NEAT is able to learn well only when the fitness estimate for each candidate policy is averaged over many episodes. Therefore, we tested a third version of Keepaway in which noise was removed from the sensors and actuators and the agents' initial states were fixed. These changes yield a domain with a completely deterministic fitness function. We hypothesized that NEAT would learn much faster in this deterministic variation as it could perfectly evaluate each network in a single episode.

To test this hypothesis, we conducted 20 trials of Sarsa and 5 trials of NEAT in the deterministic Keepaway task. Figure 5 shows the results of these experiments. In the deterministic version of the task, Sarsa's speed advantage disappears. NEAT learns more rapidly than Sarsa, in addition to discovering dramatically superior policies. Unpaired Student's t-tests confirm that the difference in average hold times is statistically significant with 95% confidence for all points with at least 10 hours of training. We conducted additional informal experiments to test whether removing the actuator noise but using a random start state, or allowing actuator noise but fixing the start state, affected the methods differently. While Sarsa performed similarly in both cases, we found that NEAT's performance improved more when actuator noise was removed than when the start state was fixed. This result makes sense since actuator noise, by affecting every state, likely adds more noise to the fitness function than randomness in the initial state.

These results confirm our hypothesis that stochasticity in the Keepaway domain critically affects how quickly NEAT can learn. In the benchmark task, NEAT learns well only when fitness estimates are averaged over many episodes, resulting in slow learning relative to Sarsa. By contrast, in the deterministic version, only one episode of evaluation is necessary for each network, enabling NEAT to learn much more rapidly. Furthermore, making the Keepaway task deterministic greatly improves, relative to Sarsa, the quality of the best policies discovered. This outcome is surprising since the deterministic version of the task is also fully observable and should be well suited to TD methods. These results demonstrate that, in the deterministic version of the task, the advantage Sarsa gains from full observability is far outweighed by the advantage NEAT gains from the ability to perform rapid and accurate fitness evaluations.

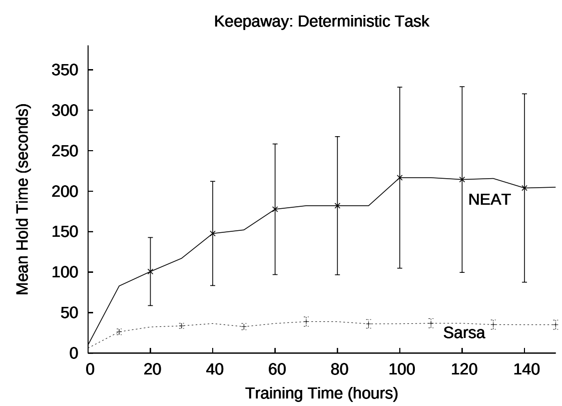


Figure 5: Mean hold times of the best policies learned by NEAT and Sarsa in the deterministic Keepaway task. Error bars show 95% confidence intervals.

Our results therefore suggest that the choice between evolutionary and TD methods can be made based on some of the target task's characteristics. In deterministic domains, where the fitness of a candidate policy can be evaluated quickly, evolutionary methods are likely to excel. If the task is fully observable but nondeterministic, TD methods may have a critical advantage. If the task is partially observable and nondeterministic, each method may have different advantages: TD methods in speed and evolutionary methods in ultimate performance.

4 On-Line Evolutionary Computation

If TD and evolutionary methods have contrasting advantages, the question naturally arises: can the performance of one method be improved by borrowing elements of the other? In our second empirical study, we demonstrate that the answer is yes. In particular, we demonstrate that the performance of evolutionary methods can be dramatically improved by borrowing exploratory mechanisms traditionally used by TD methods.

While evolutionary methods have had much empirical success in difficult RL tasks (Stanley & Miikkulainen, 2002, 2004b; Whiteson et al., 2005), this success is largely restricted to *off-line* scenarios, in which the agent learns, not in the real-world, but in a "safe" environment like a simulator. Consequently, its only goal is to learn a good policy as quickly as possible. It does not care how much reward it accrues *while it is learning* because those rewards are only hypothetical and do not correspond to real-world costs. If the agent tries disastrous policies, only computation time is lost.

Unfortunately, many reinforcement learning problems cannot be solved off-line because no simulator is available. Sometimes the dynamics of the task are unknown or are too complex to accurately simulate. In such domains, the agent has no choice but to learn *on-line*: by interacting with the real world and adjusting its policy as it goes. In an on-line learning scenario, it is not enough for an agent to learn a good policy quickly. It must also maximize the reward it accrues while it is learning because those rewards correspond to real-world costs. For example, if a robot learning on-line tries a policy that causes it to drive off a cliff, then the negative reward the agent receives is not hypothetical; it corresponds to the very real cost of fixing or replacing the robot.

To excel in on-line scenarios, a learning algorithm must effectively balance *exploration*, the search for better policies, with *exploitation*, use of the current best policy to maximize reward. Evolutionary methods already strive to perform this balance. In fact, Holland (1975) argues that the reproduction mechanism encourages exploration, since crossover and mutation result in novel genomes, but also encourages exploitation, since each new generation is based on the fittest members of the last one. However, reproduction allows evolutionary methods to balance exploration and exploitation only *across* generations, not *within* them. Once the members of each generation have been determined, they all typically receive the same evaluation time.

This approach makes sense in deterministic domains, where each member of the population can be accurately evaluated in a single episode. However, most real-world domains are stochastic, in which case fitness evaluations must be averaged over many episodes. In these domains, giving the same evaluation time to each member of the population can be grossly suboptimal because, within a generation, it is purely exploratory. Instead, an on-line evolutionary algorithm should exploit the information gained earlier in the generation to systematically give more evaluations to the most promising policies and avoid re-evaluating the weakest ones. Doing so allows evolutionary methods to increase the reward accrued during learning.

In our second empirical study, we investigate *on-line evolutionary computation*, a novel approach to achieving this balance that relies on exploratory mechanisms typically used by TD methods to balance exploration and exploitation. To use these mechanisms in evolutionary computation, we must modify the level at which they are applied. They cannot be applied at the level of individual actions because evolutionary methods, lacking value functions, have no notion of the value of individual actions. However, they can be applied at the level of episodes, in which entire policies are assessed holistically. The same exploratory mechanisms used to select individual actions in TD methods can be used to select policies for evaluation, allowing evolutionary algorithms to excel on-line by balancing exploration and exploitation within and across generations.

In particular, we examine three methods based on this approach. The first method, based on ϵ-greedy selection (Watkins, 1989), switches probabilistically between searching for better policies and re-evaluating the best known policy. Instead of distributing episodes of evaluation equally among the members of a given generation, they are assigned probabilistically. At the beginning of each episode, a policy is selected for evaluation. With probability ϵ, that policy is chosen randomly. With probability $1-\epsilon$, the current generation champion is selected. By increasing the proportion of evaluations given to the estimated strongest member of the population, ϵ-greedy evolution can boost on-line reward.

The second method, based on softmax selection (Sutton & Barto, 1998), distributes evaluations in proportion to each policy's estimated fitness. As is typical in TD methods, softmax evolution uses a Boltzmann distribution to favor policies with higher estimated fitness. Before each episode, a policy p in a population P is selected with probability

$$\frac{e^{f(p)/\tau}}{\sum_{p' \in P} e^{f(p')/\tau}} \tag{3}$$

where $f(p)$ is the fitness of policy p, averaged over all the episodes for which it has been previously evaluated. Softmax selection provides a more nuanced balance between exploration and exploitation than ϵ-greedy because it focuses its exploration on the most promising alternative to the current best policy. It can quickly abandon poorly performing policies and reduce their impact on the reward accrued during learning.

The third method, based on interval estimation (Kaelbling, 1993), computes confidence intervals for the fitness of each policy and always evaluates the policy with the highest upper bound. Hence, interval estimation evolution always selects the policy $p \in P$ that maximizes:

$$f(p) + z(\frac{100-\alpha}{200})\frac{\sigma(p)}{\sqrt{e(p)}} \tag{4}$$

where $[0, z(x)]$ is an interval within which the area under the standard normal curve is x. $f(p)$, $\sigma(p)$ and $e(p)$ are the fitness, standard deviation, and number of episodes, respectively, for policy p. Interval estimation addresses an important disadvantage of both ϵ-greedy and softmax selection: they do not consider the uncertainty of the estimates on which they base their selections. Interval estimation evolution favors policies with high estimated value and also focuses exploration on the most promising but uncertain policies.

To empirically evaluate these methods, we used two different reinforcement learning domains. The first domain, mountain car, is a standard benchmark task requiring function approximation. We use this domain to establish preliminary, proof-of-concept results. The second domain, server job scheduling, is a large, probabilistic domain drawn from the field of autonomic computing. We use this domain to assess whether these methods can scale to a much more complex task. We use these domains instead of the Keepaway domain used in the first empirical study because their simulators run much more quickly, enabling many more trials.

In the mountain car task (Sutton, 1996), an agent strives to drive a car to the top of a steep mountain. The car cannot simply accelerate forward because its engine is not powerful enough to overcome gravity. Instead, the agent must learn to drive backwards up the hill behind it, thus building up sufficient inertia to ascend to the goal.

While the mountain car task is a useful benchmark, it is a very simple domain. To assess whether evolutionary function approximation can scale to a much more complex problem, we use a challenging reinforcement learning task called server job scheduling. This domain is drawn from the burgeoning field of autonomic computing (Kephart & Chess, 2003). The goal of autonomic computing is to develop computer systems that automatically configure themselves, optimize their own behavior, or diagnose and repair their own failures.

One autonomic computing task is server job scheduling (Whiteson & Stone, 2006a), in which a server, such as a website's application server or database, must determine in what order to process the jobs currently waiting in its queue. Its goal is to maximize the aggregate utility of all the jobs it processes. A *utility function* for each job type maps the job's completion time to the utility derived by the user (Walsh, Tesauro, Kephart, & Das, 2004). Since selecting a particular job for processing necessarily delays the completion of all other jobs in the queue, the scheduler must weigh difficult trade-offs to maximize aggregate utility.

Our experiments were conducted in a Java-based simulator. During each timestep, the server removes one job from its queue and completes it. Also, a new job of a randomly selected type is added to the end of the queue. Hence, the agent must make decisions about which job to process next even as new jobs are arriving. For each job that completes, the scheduling agent receives an immediate reward determined by that job's utility function. Utility functions for the four job types used in our experiments are shown in Figure 6. These utility functions result in a challenging scheduling problem because they are 1) non-linear and 2) different for each job type. Hence, the cost of delaying completion of any particular job depends not only on its type but on its current age. Note that all these utilities are negative functions of completion time. Hence, the scheduling agent strives to bring aggregate utility as close to zero as possible.

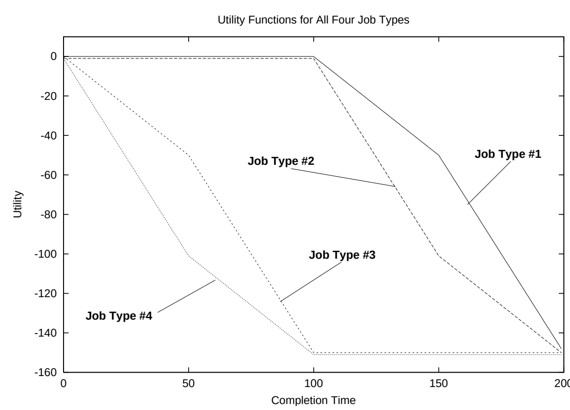
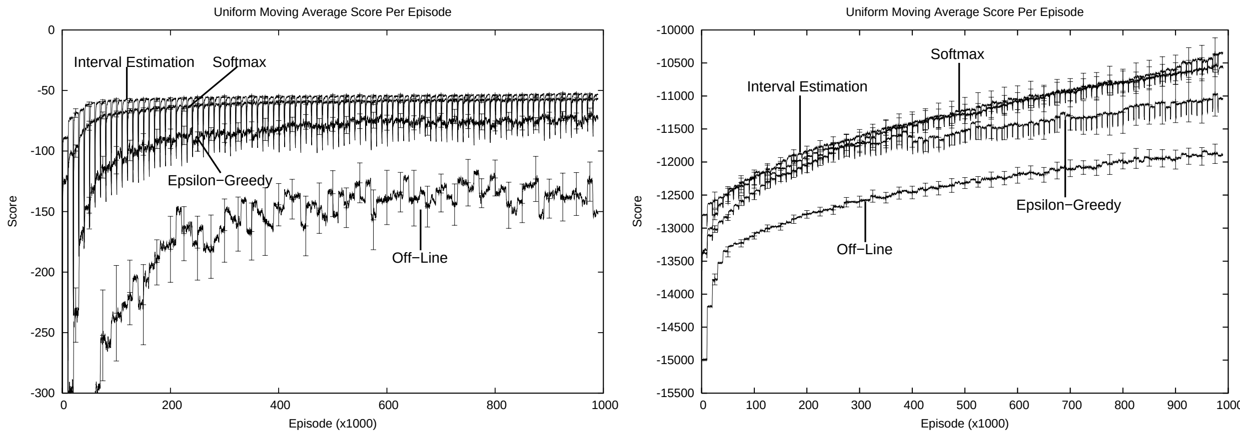


Figure 6: The four utility functions.

As a baseline of comparison, we applied the original, off-line version of NEAT to both the mountain car and server job scheduling domains and averaged its performance over 25 runs. NEAT parameters were set the same as reported in previous work (Whiteson & Stone, 2006a). The population size $|P|$ was 100 and the number of episodes per generation e was 10,000. Hence, each member of the population was evaluated for 100 episodes. Next, we applied the ϵ-greedy, softmax, and interval estimation versions of NEAT to both domains using the same parameter settings. Each of these on-line methods has associated with it one additional parameter which controls the balance between exploration and exploitation. For each method, we experimented informally with approximately ten different settings of these parameters to find ones that worked well in the two tasks. Finally, we averaged the performance of each method over 25 runs using the best known parameter settings.

Those settings were as follows. For ϵ-greedy, ϵ was set to 0.25. This value is larger than is typically used in TD methods but makes intuitive sense, since exploration in NEAT is safer than in TD methods. After all, even when NEAT explores, the policies it selects are not drawn randomly from policy space. On the contrary, they are the children of the previous generation's fittest parents. For softmax, the appropriate value of τ depends on the range of fitness scores, which



(a) Mountain Car (b) Server Job Scheduling

Figure 7: The uniform moving average reward accrued by off-line NEAT, compared to three versions of on-line NEAT in the mountain car and server job scheduling domains.

differs dramatically between the two domains. Hence, different values were required for the two domains: we set τ to 50 in mountain car and 500 in server job scheduling. For interval estimation, α was set to 20, resulting in 80% confidence intervals.

Figure 7 summarizes the results of these experiments by plotting a uniform moving average over the last 1,000 episodes of the total reward accrued per episode for each method. We plot average reward because it is an on-line metric: it measures the amount of reward the agent accrues while it is learning. The best policies discovered by evolution, i.e. the generation champions, perform substantially higher than this average. However, using their performance as an evaluation metric would ignore the on-line cost that was incurred by evaluating the rest of population and receiving less reward per episode. Error bars on the graph indicate 95% confidence intervals. In addition, unpaired Student's t-tests confirm, with 95% confidence, the statistical significance of the performance difference between each pair of methods except softmax and interval estimation.

The results clearly demonstrate that exploratory mechanisms borrowed from TD methods can dramatically improve the on-line performance of evolutionary computation. All three on-line methods substantially outperform the off-line version of NEAT. In addition, the more nuanced strategies of softmax and interval estimation fare better than ϵ-greedy. This result is not surprising since the ϵ-greedy approach simply interleaves the search for better policies with exploitative episodes that employ the best known policy. Softmax selection and interval estimation, by contrast, concentrate exploration on the most promising alternatives. Hence, they spend fewer episodes on the weakest individuals and achieve better performance as a result.

The on-line methods, especially interval estimation, show a series of periodic fluctuations. Each of these fluctuations corresponds to one generation. The performance improvements within each generation reflect the on-line methods' ability to exploit the information gleaned from earlier episodes. As the generation progresses, these methods become better informed about which individuals to favor when exploiting and average reward increases as a result.

5 Evolutionary Function Approximation

The empirical study presented above demonstrates that evolution can be substantially improved by incorporating elements of TD methods. However, the resulting algorithm remains essentially an evolutionary one. In our third empirical study, we investigate a more ambitious integration of evolutionary and TD methods. We show that the resulting combination, which we call *evolutionary function approximation*, outperforms each of its constituent components.

When evolutionary methods are applied to reinforcement learning problems, they typically evolve a population of action selectors, each of which remains fixed during its fitness evaluation. The central insight behind evolutionary function approximation is that, if evolution is directed to evolve value functions instead, then those value functions can be updated, using TD methods, during each fitness evaluation. In this way, the system can *evolve* function approximators that are better able to *learn* via TD.

By automatically finding effective representations, evolution can improve the performance of TD methods. However, TD methods can also improve the performance of evolution. In particular, evolutionary function approximation can enable synergistic effects between evolution and learning via a biological phenomenon called the *Baldwin Effect* (Baldwin, 1896). This effect, which has been demonstrated in evolutionary computation (Hinton & Nowlan, 1987; Ackley & Littman, 1991), occurs in two stages. First, learning speeds evolution because each individual need not be perfect at birth; it need only be in the right neighborhood and learning adjusts it accordingly. Second, those behaviors that were previously learned become known at birth because individuals that possess them at birth have higher overall fitness and are favored by evolution.

In our third empirical study, we investigate the performance of NEAT+Q, an implementation of evolutionary function approximation that uses NEAT to evolve topologies and initial weights of neural networks that are better able to learn, via backpropagation, to represent the value estimates provided by Q-learning. All that is required to make NEAT optimize value functions instead of action selectors is a reinterpretation of its output values. The structure of neural network action selectors (one input for each state feature and one output for each action) is already identical to that of Q-learning function approximators. Therefore, if the weights of the networks NEAT evolves are updated during their fitness evaluations using Q-learning and backpropagation, they will effectively evolve value functions instead of action selectors. Hence, the outputs are no longer arbitrary values; they represent the long-term discounted values of the associated state-action pairs and are used, not just to select the most desirable action, but to update the estimates of other state-action pairs.

NEAT+Q combines the power of TD methods with the ability of NEAT to learn effective representations. Traditional neural network function approximators put all their eggs in one basket by relying on a single manually designed network to represent the value function. NEAT+Q, by contrast, explores the space of such networks to increase the chance of finding a representation that will perform well.

To evaluate evolutionary function approximation, we tested it in the mountain car and server job scheduling domains (Whiteson & Stone, 2006a). Mountain car is a particularly relevant domain because it is known to be difficult for neural network function approximators. Previous research has demonstrated that TD methods can solve the task using several different function approximators, including CMACs (Sutton, 1996), locally weighted regression (Boyan & Moore, 1995), and radial basis functions (Kretchmar & Anderson, 1997). By giving the learner *a priori* knowledge about the goal state and using methods based on experience replay, the mountain car problem has been solved with neural networks too (Reidmiller, 2005). However, the task remains notoriously problematic for neural networks, as several researchers have noted that value estimates can easily diverge (e.g. Boyan & Moore, 1995; Pyeatt & Howe, 2001). We hypothesized that this difficulty is due at least in part to the problem of finding an appropriate representation. We use the mountain car domain as a preliminary testbed for NEAT+Q in order to evaluate this hypothesis.

We tested NEAT+Q by conducting 25 runs in each domain and comparing the results to regular NEAT (i.e. the off-line NEAT results presented in Section 4). NEAT+Q used the same parameter settings. In addition, the eligibility decay rate λ was 0.0, the learning rate α was set to 0.1, and α was annealed linearly for each member of the population until reaching zero after 100 episodes.[3] In scheduling, the discount factor γ was 0.95 and the agents use ϵ-greedy exploration with $\epsilon = 0.05$. Those values of γ and ϵ work well in mountain car too, though in the experiments presented here they were set to 1.0 and 0.0 respectively, since Sutton (1996) found that discounting and exploration are unnecessary in mountain car.

To test Q-learning without NEAT, we tried 24 different manual configurations of the neural network function approximator in each domain. These configurations correspond to every possible combination of the following parameter settings. The networks had feed-forward topologies with 0, 4, or 8 hidden nodes. The learning rate α was either 0.01 or 0.001. The annealing schedules for α were linear, decaying to zero after either 100,000 or 250,000 episodes. The eligibility decay rate λ was either 0.0 or 0.6. The other parameters, γ and ϵ, were set just as with NEAT+Q, and the standard deviation of initial weights σ was 0.1. Each of these 24 configurations was evaluated for 5 runs. In addition, we experimented informally with higher and lower values of α, higher values of γ, slower linear annealing, exponential annealing, and no annealing at all, though none performed as well as the results presented here. We took the setting with the highest performance[4] and conducted an additional 20 runs, for a total of 25.

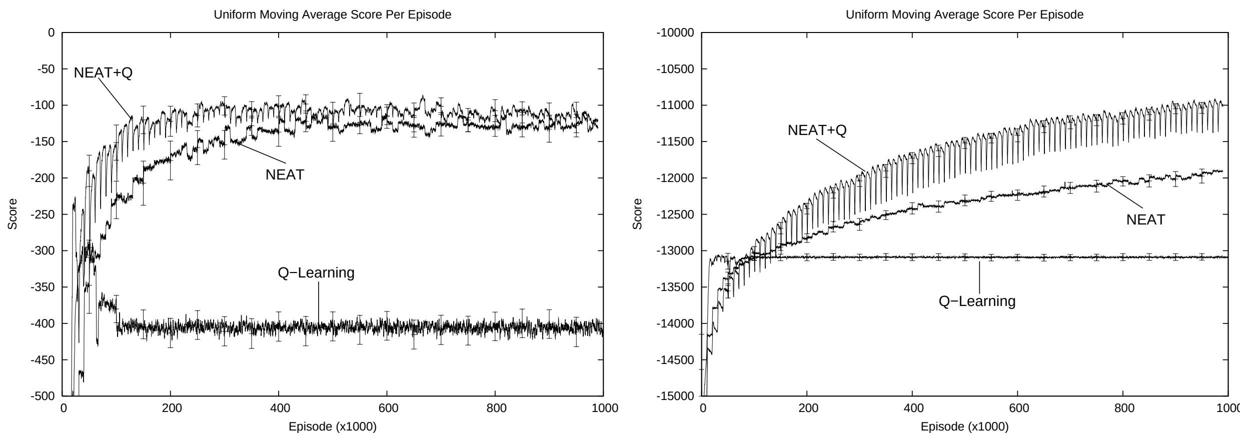
Figure 8 shows the results of our experiments in both domains comparing NEAT+Q to NEAT and Q-learning with manually designed networks. Only the highest performing Q-learning configuration is shown. For each method, the corresponding line in the graph represents a uniform moving average over the aggregate reward received in the past 1,000 episodes. Error bars indicate 95% confidence intervals. In addition, unpaired Student's t-tests confirmed the statistical significance of the performance difference between each pair of methods.

Note that, as with the on-line methods presented in Section 4, the progress of NEAT+Q consists of a series of periodic fluctuations. Each period corresponds to one generation and the changes within them are due to learning via backpropagation. Though each individual learns for many episodes, those episodes do not occur consecutively but are spread across the entire generation. Hence, each individual changes gradually during the generation as it is repeatedly evaluated. The result is a series of intra-generational learning curves within the larger learning curve.

For the particular problems we tested and network configurations we tried, evolutionary function approximation significantly improves performance over manually designed networks. NEAT+Q also significantly outperforms regular NEAT in both domains. This result highlights the value of TD methods on challenging reinforcement learning problems. Even when NEAT is employed to find effective representations, the best performance is achieved only when TD methods are used to estimate a value function. Hence, the relatively poor performance of Q-learning is not due to some weakness in the TD methodology but merely to the failure to find a good representation.

[3] Other values of λ were tested in the context of NEAT+Q but had little effect on performance.

[4] Mountain car parameters were: 4 hidden nodes, $\alpha = 0.001$, annealed to zero at episode 100,000, $\lambda = 0.0$. Server job scheduling parameters were: 4 hidden nodes, $\alpha = 0.01$, annealed to zero at episode 100,000, $\lambda = 0.6$.



(a) Mountain Car (b) Server Job Scheduling

Figure 8: A comparison of the performance of manual and evolutionary function approximators in the mountain car and server job scheduling domains.

Nonetheless, the possibility remains that additional engineering of the network structure, the feature set, or the learning parameters would significantly improve Q-learning's performance. In particular, when Q-learning is started with one of the best networks discovered by NEAT+Q and the learning rate is annealed aggressively, Q-learning matches NEAT+Q's performance without directly using evolutionary computation. However, it is unlikely that a manual search, no matter how extensive, would discover these successful topologies, which contain irregular and partially connected hidden layers.

In mountain car, the performance of the final NEAT+Q generation champions matches the best results published by other researchers (e.g. Smart & Kaelbling, 2000). However, this does not imply that neural networks are the function approximator of choice for the mountain car domain. On the contrary, linear function approximators can solve this task in many fewer episodes (Sutton, 1996). This is not surprising because NEAT+Q is actually solving a more challenging problem. Linear function approximators require the human designer to engineer a state representation in which the optimal value function is linear with respect to those state features (or can be reasonably approximated as such). By contrast, nonlinear function approximators like neural networks can take a simpler state representation and *learn* the important nonlinear relationships. However, doing so with neural networks is notoriously difficult in the mountain car task. Our results demonstrate that evolutionary function approximation can overcome these difficulties.

Comparing Figures 7 and 8 reveals that on-line evolutionary computation provides a greater performance boost than evolutionary function approximation. However, it is not necessary to choose between these two approaches as they can be easily combined. Additional experiments (Whiteson & Stone, 2006a) demonstrate that on-line evolutionary computation can boost the performance of NEAT+Q as well as NEAT.

6 Related Work

A broad range of previous research is related in terms of both methods and goals to the techniques presented in this article. This section highlights some of that research and contrasts it with this work.

6.1 Optimizing Representations for TD Methods

A major challenge of using TD methods is finding good representations for function approximators. This article addresses that problem by coupling TD methods with evolutionary techniques like NEAT that are proven representation optimizers. However, many other approaches are also possible.

One strategy is to train the function approximator using supervised methods that also optimize representations. For example, Rivest and Precup (2003) train cascade-correlation networks as TD function approximators. Cascade-correlation networks are similar to NEAT in that they grow internal topologies for neural networks. However, instead of using evolutionary computation to find such topologies, they rely on the network's error on a given training set to compare alternative representations. The primary complication of Rivest and Precup's approach is that cascade-correlation networks, like many representation-optimizing supervised methods, need a large and stable training set, which TD methods do not provide. Rivest and Precup address this problem with a caching system that in effect creates a hybrid value function consisting of a table and a neural network. While this approach delays the exploitation of the agent's experience, it nonetheless represents a promising way to marry the representation-optimizing capacity of cascade-correlation networks and other supervised algorithms with the power of TD methods.

Mahadevan (2005) suggests another strategy: using spectral analysis to derive basis functions for TD function approximators. His approach is similar to this work in that the agent is responsible for learning both the value function and its representation. It is different in that the representation is selected by analyzing the underlying structural properties of the state space, rather than evaluating potential representations in the domain.

A third approach is advanced by Sherstov and Stone (2005): using the Bellman error generated by TD updates to assess the reliability of the function approximator in a given region of the state or action space. They use this metric to automatically adjust the breadth of generalization for a CMAC function approximator. An advantage of this approach is that feedback arrives immediately, since Bellman error can be computed after each update. A disadvantage is that the function approximator's representation is not selected based on its actual performance, which may correlate poorly with Bellman error.

6.2 Combining Evolution with Other Learning Methods

Because of the potential performance gains offered by the Baldwin Effect, many researchers have developed methods that combine evolutionary computation with other learning methods that act within an individual's lifetime. Some of this work is applied to supervised problems, in which evolutionary computation can be coupled with any supervised learning technique such as backpropagation in a straightforward manner. For example, Boers et al. (1995) introduce a neuroevolution technique that, like NEAT, tries to discover appropriate topologies. They combine this method with backpropagation and apply the result to a simple supervised learning problem. Also, Giraud-Carrier (2000) uses a genetic algorithm to tune the parameters of RBF networks, which he applies to a supervised classification problem.

Inducing the Baldwin Effect on reinforcement learning problems is more challenging, since they do not automatically provide the target values necessary for supervised learning. Evolutionary function approximation uses TD methods to estimate those targets, though researchers have tried many other approaches. McQuestion and Miikkulainen (1997) present a neuroevolutionary technique that relies on each individual's parents to supply targets and uses backpropagation to train towards those targets. Stanley et al. (2003) avoid the problem of generating targets by using Hebbian rules, an unsupervised technique, to change a neural network during its fitness evaluation. Downing (2001) combines genetic programming with Q-learning using a simple tabular representation; genetic programming automatically learns how to discretize the state space.

Nolfi et al. (1994) present a neuroevolutionary system that adds extra outputs to the network that are designed to predict what inputs will be presented next. When those inputs actually arrive, they serve as targets for backpropagation, which adjusts the network's weights starting from the added outputs. This technique allows a network to be adjusted during its lifetime using supervised methods but relies on the assumption that forcing it to learn to predict future inputs will help it select appropriate values for the remaining outputs, which actually control the agent's behavior. Another significant restriction is that the weights connecting hidden nodes to the action outputs cannot be adjusted at all during each fitness evaluation.

Ackley and Littman (1991) combine neuroevolution with reinforcement learning in an artificial life context. Evolutionary computation optimizes the initial weights of an "action network" that controls an agent in a foraging scenario. The weights of the network are updated during each individual's lifetime using a reinforcement learning algorithm called CRBP on the basis of a feedback signal that is also optimized with neuroevolution. Hence, their approach is similar to the one described in this article, though the neuroevolution technique they employ does not optimize network topologies and CRBP does not learn a value function.

Another important related method is VAPS (Baird & Moore, 1999). While it does not use evolutionary computation, it does combine TD methods with policy search methods. It provides a unified approach to reinforcement learning that uses gradient descent to try to simultaneously maximize reward and minimize error on Bellman residuals. A single parameter determines the relative weight of these goals. Because it integrates policy search and TD methods, VAPS is in much the same spirit as evolutionary function approximation. However, the resulting methods are quite different. While VAPS provides several impressive convergence guarantees, it does not address the question of how to represent the value function.

Other researchers have also sought to combine TD and policy search methods. For example, Sutton et al. (2000a) use policy gradient methods to search policy space but rely on TD methods to obtain an unbiased estimate of the gradient. Similarly, in actor-critic methods (Konda & Tsitsiklis, 1999), the actor optimizes a parameterized policy by following a gradient informed by the critic's estimate of the value function. Like VAPS, these methods do not learn a representation for the value function.

6.3 Variable Evaluations in Evolutionary Computation

Because it allows members of the same population to receive different numbers of evaluations, the approach to on-line evolution presented here is similar to previous research about optimizing noisy fitness functions. For example, Stagge (1998) introduces mechanisms for deciding which individuals need more evaluations for the special case where the noise is Gaussian. Beielstein and Markon (2002) use a similar approach to develop tests for determining which individuals should survive. However, this area of research has a significantly different focus, since the goal is to find the best individuals using the fewest evaluations, not to maximize the reward accrued during those evaluations.

The problem of using evolutionary systems on-line is more closely related to other research about the exploration/exploitation tradeoff, which has been studied extensively in the context of TD methods (Watkins, 1989; Sutton & Barto, 1998) and multiarmed bandit problems (Bellman, 1956; Macready & Wolpert, 1998; Auer, Cesa-Bianchi, & Fischer, 2002). The selection mechanisms we employ in our system are well-established though, to our knowledge, their application to evolutionary computation is novel.

7 Ongoing and Future Work

There are many ways that the work presented in this article could be extended, refined, or further evaluated. This section enumerates a few of the possibilities.

Reducing Sample Complexity One disadvantage of evolutionary function approximation is its high sample complexity, since each fitness evaluation lasts for many episodes. However, in domains where the fitness function is not too noisy, each fitness evaluation can be conducted in a single episode if the candidate function approximator is pre-trained using methods based on experience replay (Lin, 1992). By saving sample transitions from the previous generation, each new generation can be be trained off-line. This variation uses much more computation time but many fewer sample episodes. Since sample experience is typically a much scarcer resource than computation time, this enhancement can greatly improve the practical applicability of evolutionary function approximation. In other work, we show that a sample-efficient version of NEAT+Q can perform as well regular NEAT+Q but with dramatically lower sample complexity (Whiteson & Stone, 2006b).

Analyzing Evolved Topologies Preliminary efforts analyzing networks generated by NEAT+Q demonstrate that it tends to learn sparsely connected networks with a small number of hidden nodes (Whiteson & Stone, 2006a). The topologies are typically quite idiosyncratic, unlike those a human would likely design. In the future, we hope to study these networks further to determine, for example, whether the hidden nodes represent useful state abstractions. Such analysis could produce important insights into what properties of neural networks are critical for successful TD function approximation.

Using Different Policy Search Methods This article focuses on using evolutionary methods to automate the search for good function approximator representations. However, many other forms of policy search such as PEGASUS (Ng & Jordan, 2000) and policy gradient methods (Sutton, McAllester, Singh, & Mansour, 2000b; Kohl & Stone, 2004) have also succeeded on difficult reinforcement learning tasks. TD methods could be combined with these methods in the same way they are combined with evolutionary computation in this article. In the future, we plan to test some of these alternative combinations.

8 Conclusion

This article presents three empirical studies that compare, contrast, and combine evolutionary and TD methods for action selection with reinforcement learning. The first study, which compares these two approaches in Keepaway, demonstrates that evolutionary approaches can outperform TD methods in difficult RL tasks. It also highlights the contrasting strengths of each approach, as TD methods perform better in the Markovian version and evolutionary methods perform better in the deterministic version. The second study, which investigates on-line evolution, demonstrates that evolutionary methods, by borrowing mechanisms traditionally used in TD methods, can improve the way they balance exploration and exploitation. As a result, they can excel at on-line tasks. The third study, which investigates evolutionary function approximation, demonstrates that human designers do not necessarily have to choose between these two approaches. On the contrary, they can combine them synergistically, yielding a new approach that performs better than either approach by itself. Together these results suggest that a rich and promising research area lies at the intersection of these two very different approaches to action selection in reinforcement learning.

Acknowledgments

Thanks to Richard Sutton, Michael Littman, Gerry Tesauro, and Manuela Veloso for helpful discussions and ideas. Thanks to Risto Miikkulainen, Nick Jong, Bikram Banerjee, Shivaram Kalyanakrishnan, Nate Kohl, David Pardoe, Joeseph Reisinger, Jefferson Provost, Ken Stanley and the anonymous reviewers for constructive comments about earlier versions of this work. This research was supported in part by NSF CAREER award IIS-0237699, NSF award EIA-0303609, DARPA grant HR0011-04-1-0035, and an IBM faculty award.

References

Ackley, D., & Littman, M. (1991). Interactions between learning and evolution. *Artificial Life II, SFI Studies in the Sciences of Complexity*, *10*, 487–509.

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47(2-3)*, 235–256.

Baird, L., & Moore, A. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. MIT Press.

Baldwin, J. M. (1896). A new factor in evolution. *The American Naturalist*, *30*, 441–451.

Beielstein, T., & Markon, S. (2002). Threshold selection, hypothesis tests and DOE methods. In *2002 Congresss on Evolutionary Computation*, pp. 777–782.

Bellman, R. E. (1956). A problem in the sequential design of experiments. *Sankhya*, *16*, 221–229.

Boers, E., Borst, M., & Sprinkhuizen-Kuyper, I. (1995). Evolving Artificial Neural Networks using the "Baldwin Effect". Tech. rep. TR 95-14.

Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*.

Crites, R. H., & Barto, A. G. (1998). Elevator group control using multiple reinforcement learning agents. *Machine Learning*, *33*(2-3), 235–262.

Downing, K. L. (2001). Reinforced genetic programming. *Genetic Programming and Evolvable Machines*, *2(3)*, 259–288.

Giraud-Carrier, C. (2000). Unifying learning with evolution through Baldwinian evolution and Lamarckism: A case study. In *Proceedings of the Symposium on Computational Intelligence and Learning (CoIL-2000)*, pp. 36–41. MIT GmbH.

Gomez, F., & Miikkulainen, R. (2002). Robust nonlinear control through neuroevolution. Tech. rep. AI02-292.

Gomez, F., & Schmidhuber, J. (2005). Co-evolving recurrent neurons learn deep memory POMDPs. In *GECCO-05: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 491–498.

Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, *1*, 495–502.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Hsu, W. H., & Gustafson, S. M. (2002). Genetic programming and multi-agent layered learning by reinforcements. In *Genetic and Evolutionary Computation Conference*, pp. 764–771.

Kaelbling, L. P. (1993). *Learning in Embedded System*. MIT Press.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36(1)*, 41–50.

Kohl, N., & Stone, P. (2004). Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pp. 611–616.

Konda, V. R., & Tsitsiklis, J. N. (1999). Actor-critic algorithms. In *Advances in Neural Information Processing Systems 11*, pp. 1008–1014.

Kostiadis, K., & Hu, H. (2001). KaBaGe-RL: Kanerva-based generalisation and reinforcement learning for possession football. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*.

Kretchmar, R. M., & Anderson, C. W. (1997). Comparison of CMACs and radial basis functions for local function approximators in reinforcement learning. In *International Conference on Neural Networks*.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, *8(3-4)*, 293–321.

Macready, W. G., & Wolpert, D. H. (1998). Bandit problems and the exploration/exploitation tradeoff. In *IEEE Transactions on Evolutionary Computation*, Vol. 2(1), pp. 2–22.

Mahadevan, S. (2005). Samuel meets Amarel: Automating value function approximation using global state space analysis. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

McQuesten, P., & Miikkulainen, R. (1997). Culling and teaching in neuro-evolution. In Bäck, T. (Ed.), *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 760–767.

Moriarty, D. E., & Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, *22*, 11–32.

Moriarty, D. E., Schultz, A. C., & Grefenstette, J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, *11*, 241–276.

Ng, A. Y., & Jordan, M. I. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pp. 406–415. Morgan Kaufmann Publishers Inc.

Nolfi, S., Elman, J. L., & Parisi, D. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, *2*, 5–28.

Pollack, J. B., Blair, A. D., & Land, M. (1997). Coevolution of a backgammon player. In Langton, C. G., & Shimohara, K. (Eds.), *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems*, pp. 92–98 Cambridge, MA. The MIT Press.

Powell, M. J. D. (1987). Radial basis functions for multivariate interpolation: A review. In Mason, J. C., & Cox, M. G. (Eds.), *Algorithms for Approximation*, pp. 143–167 Oxford. Clarendon Press.

Prescott, T. J., Bryson, J. J., & Seth, A. K. (2007). Modelling natural action selection: An introduction to the theme issue. *Philosophical Transactions of the Royal Society - B*. To appear.

Pyeatt, L. D., & Howe, A. E. (2001). Decision tree function approximation in reinforcement learning. In *Proceedings of the Third International Symposium on Adaptive Systems: Evolutionary Computation and Probabilistic Graphical Models*, pp. 70–77.

Reidmiller, M. (2005). Neural fitted Q iteration - first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pp. 317–328.

Rivest, F., & Precup, D. (2003). Combining TD-learning with cascade-correlation networks. In *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 632–639. AAAI Press.

Rummery, G. A., & Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University.

Runarsson, T. P., & Lucas, S. M. (2005). Co-evolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation*, *9*, 628–640.

Sherstov, A. A., & Stone, P. (2005). Function approximation via tile coding: Automating parameter choice. In Zucker, J.-D., & Saitta, I. (Eds.), *SARA 2005*, Vol. 3607 of *Lecture Notes in Artificial Intelligence*, pp. 194–205. Springer Verlag, Berlin.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replaceing eligibility traces. *Machine Learning*, *22*, 123–158.

Smart, W. D., & Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the Seventeeth International Conference on Machine Learning*, pp. 903–910.

Stagge, P. (1998). Averaging efficiently in the presence of noise. In *Parallel Problem Solving from Nature*, Vol. 5, pp. 188–197.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2003). Evolving adaptive neural networks with and without adaptive synapses. In *Proceeedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, Vol. 4, pp. 2557–2564.

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

Stanley, K. O., & Miikkulainen, R. (2004a). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, *21*, 63–100.

Stanley, K. O., & Miikkulainen, R. (2004b). Evolving a roving eye for go. In *Proceedinngs of the Genetic and Evolutionary Computation Conference*.

Stone, P., Kuhlmann, G., Taylor, M. E., & Liu, Y. (2006). Keepaway soccer: From machine learning testbed to benchmark. In Noda, I., Jacoff, A., Bredenfeld, A., & Takahashi, Y. (Eds.), *RoboCup-2005: Robot Soccer World Cup IX*, Vol. 4020, pp. 93–105. Springer Verlag, Berlin.

Stone, P., Sutton, R. S., & Kuhlmann, G. (2005). Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, *13*(3), 165–188.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, pp. 1038–1044.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000a). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., & Muller, K.-R. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 12, pp. 1057–1063. The MIT Press.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (2000b). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pp. 1057–1063.

Taylor, M. E., Whiteson, S., & Stone, P. (2006). Comparing evolutionary and temporal difference methods in a reinforcement learning domain. In *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1321–1328.

Walsh, W. E., Tesauro, G., Kephart, J. O., & Das, R. (2004). Utility functions in autonomic systems. In *Proceedings of the International Conference on Autonomic Computing*, pp. 70–77.

Watkins, C. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge.

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

Whiteson, S., & Stone, P. (2006a). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, *7(May)*, 877–917.

Whiteson, S., & Stone, P. (2006b). Sample-efficient evolutionary function approximation for reinforcement learning. In *AAAI 2006: Twenty-First National Conference on Artificial Intelligence*, pp. 518–523.

Whitley, D., Dominic, S., Das, R., & Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, *13*, 259–284.

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