A Complexity Analysis of Cooperative Mechanisms in Reinforcement Learning

Steven D. Whitehead
Department of Computer Science
University of Rochester
Rochester, NY 14627
email: white@cs.rochester.edu

Abstract

Reinforcement learning algorithms, when used to solve multi-stage decision problems, perform a kind of online (incremental) search to find an optimal decision policy. The time complexity of this search strongly depends upon the size and structure of the state space and upon *a priori* knowledge encoded in the learners initial parameter values. When *a priori* knowledge is not available, search is unbiased and can be excessive.

Cooperative mechanisms help reduce search by providing the learner with shorter latency feedback and auxiliary sources of experience. These mechanisms are based on the observation that in nature, intelligent agents exist in a cooperative social environment that helps structure and guide learning. Within this context, learning involves information transfer as much as it does discovery by trial-and-error.

Two cooperative mechanisms are described: *Learning with an External Critic* (or LEC) and *Learning By Watching* (or LBW). The search time complexity of these algorithms, along with unbiased Q-learning, are analyzed for problem solving tasks on a restricted class of state spaces. The results indicate that while unbiased search can be expected to require time moderately exponential in the size of the state space, the LEC and LBW algorithms require at most time linear in the size of the state space and under appropriate conditions, are independent of the state space size altogether; requiring time proportional to the length of the optimal solution path. While these analytic results apply only to a restricted class of tasks, they shed light on the complexity of search in reinforcement learning in general and the utility of cooperative mechanisms for reducing search.

Introduction

When reinforcement learning is used to solve multi-stage decision problems, learning can be viewed as a search process in which the agent, by executing a sequence of actions, searches the world for states that yield reward. For real-world tasks, the state space may be large and rewards may be sparse. Under these circumstances the time required to learn a control policy may be excessive. The detrimental effects of search manifest themselves most at the beginning of the task when the agent has an initially unbiased control strategy, and in the middle of a task when changes occur in the environment that invalidate an existing control policy.

Two cooperative learning algorithms are proposed to reduce search and decouple the learning rate from state-space size. The first algorithm, called *Learning with an External Critic* (or LEC), is based on the idea of a mentor, who watches the learner and generates immediate rewards in response to its most recent actions. This reward is then used temporarily to bias the learner's control strategy. The second algorithm, called *Learning By Watching* (or LBW), is based on the idea that an agent can gain experience vicariously by relating the observed behavior of others to its own. While LEC algorithms require interaction with knowledgeable agents, LBW algorithms can be effective even when interaction is with equally naive peers.

The principle idea being advocated in both LEC and LBW is that, in nature, intelligent agents do not exist in isolation, but are embedded in a benevolent society that is used to guide and structure learning. Humans learn by watching others, by being told, and by receiving criticism and encouragement. *Learning is more often a transfer than a discovery.* Similarly, intelligent robots cannot be expected to learn complex real-world tasks in isolation by trial-and-error alone. Instead, they must be embedded in cooperative environments, and algorithms must be developed to facilitate the transfer of knowledge among them. Within this context, trial-and-error learning continues to play a crucial role: for pure discovery purposes and for refining and elaborating knowledge acquired from others.

The search time complexity is analyzed for pure unbiased Q-learning, LEC, and LBW algorithms for an important class of state spaces. Generally, the results

indicate that unbiased Q-learning can have a search time that is exponential in the depth of the state space, while the LEC and LBW algorithms require at most time linear in the state space size and under appropriate conditions, time independent of the state space size and proportional to the length of the optimal solution path.

In the analysis that follows, only definitions and theorems are given. Proofs can be found in the appendix.

Characterizing state spaces

Naturally, the scaling properties of any reinforcement learning algorithm strongly depends upon the structure of the state space and the details of the algorithm itself. Therefore, it is difficult to obtain results for completely general situations. However, by making some simplifications we can obtain interesting results for a representative class of tasks and we can gain insight into more general situations. Below we define a number of properties that are useful when talking about classes of state spaces. We assume that actions are deterministic.

Definition 1 (1-step invertible) *A state space is* 1-step invertible *if every action has an inverse. That is, if in state x, action a causes the system to enter state y, there exists an action a^{-1} that when executed in state y causes the system to enter state x.*[1]

Definition 2 (uniformly k-bounded) *A state space is* uniformly k-bounded *with respect to a state x if*

1. *The maximum number of steps needed to reach x from anywhere in the state space is k.*
2. *All states whose distance to x is less than k have b_- actions that decrease the distance to x by one, b_+ actions that increase the distance to x by one, and $b_=$ actions that leave the distance to x unchanged.*
3. *all states whose distance to x is k have b_- actions that decrease the distance by one and $b_= + b_+$ actions that leave the distance unchanged.*[2]

Definition 3 (homogeneous) *A state space is* homogeneous *with respect to state x if it is* 1-step invertible *and* uniformly k-bounded *with respect to x.*

Definition 4 (polynomial width) *A homogeneous state space (of depth k) has* polynomial width *if the size of the state space is a polynomial function of its depth (k).*

For example, 2 and 3 dimensional grids have polynomial width since the size of their state spaces scale as $O(k^2)$ and $O(k^3)$ respectively.

[1] k-step invertibility can be defined analogously and results similar to those described below can be obtained.

[2] That is, at the boundaries, actions that would normally increase the distance to x are folded into actions the leave the distance unchanged.

Homogeneous state spaces are useful for studying the scaling properties of reinforcement learning algorithms because they are analytically tractable. They represent an idealization of the state spaces typically studied in AI — in particular, the boundaries of the state space are smooth and equally distant from the "center" state x, and the interior states share the same local connectivity pattern. Nevertheless, we expect the complexity result obtained for homogeneous state spaces to be indicative of the scaling properties of more general state spaces.

Q-learning Analysis

To study the time complexity of unbiased systems, we have chosen to analyze Q-learning [Watkins, 1989] as a representative reinforcement learning algorithm. Although a variety of other reinforcement algorithms have been described in the literature [Barto *et al.*, 1983; Holland *et al.*, 1986; Watkins, 1989; Jordan and Rumelhart, 1990], most are similar to Q-learning in that they use temporal difference methods [Sutton, 1988] to estimate a utility function that is used to determine the system's decision policy. Thus, even though our analysis is for Q-learning, we expect our results to apply to other algorithms as well.

In Q-learning, the system's objective is to learn a control policy π, which maps states into actions, that maximizes the discounted cumulative reward:

$$\mathbf{r}_t = \sum_{n=0}^{\infty} \gamma^n r_{t+n} \qquad (1)$$

where $\mathbf{r}_t$ is the discounted cumulative reward (also called the *return*), γ is the discount rate ($0 \leq \gamma < 1$), and r_t is the reward received after executing an action at time t.

The system maintains an *action-value* function, Q, that maps state-action pairs into expected returns. That is, $Q(x, a)$ is the system's *estimate* of the return it expects to receive given that it executes action a in state x and follows the optimal policy thereafter. Given Q the system's policy, π is determined by the rule:

$$\pi(x) = a \text{ such that } Q(x,a) = \max_{b \in A}[Q(x,b)], \qquad (2)$$

where A is the set of possible actions.

The estimates represented by the action-value function are incrementally improved through trial-and-error by using updating rules based on temporal difference (TD) methods [Sutton, 1988]. In 1-step Q-learning only the action-value of the most recent state-action pair is updated. The rule used is

$$Q(x_t, a_t) \leftarrow (1-\alpha)Q(x_t, a_t) + \alpha[r_t + \gamma U(x_{t+1})], \qquad (3)$$

where

$$U(x) = \max_{b \in A}[Q(x,b)], \qquad (4)$$

where x_t, a_t, r_t are the state, action, and reward at time t respectively, and where α is the learning rate parameter.

1. $x \leftarrow$ the current state
2. Select an action a that is usually consistent with $\pi(x)$, but occasionally an alternate. For example, one might choose a according to the Boltzmann distribution: $p(a|x) = \frac{e^{Q(x,a)/T}}{\sum_{b \in A} e^{Q(x,b)/T}}$ where T is a temperature parameter that adjusts the degree of randomness.
3. Execute action a, and let y be the next state and r the reward received.
4. Update the action-value function:

$$Q(x,a) \leftarrow (1-\alpha)Q(x,a) + \alpha[r + \gamma U(y)],$$

where $U(y) = \max_{b \in A}[Q(y,b)]$.
5. Go to 1.

Figure 1: The 1-step Q-learning algorithm

Other Q-learning algorithms use different rules for updating the action-value function. Figure 1 summarizes the steps for 1-step Q-learning.

Definition 5 (zero-initialized) *A Q-learning system is* zero initialized *if all its action-values are initially zero.*

Definition 6 (problem solving task) *A* problem solving task *is defined as any learning task where the system receives a reward only upon entering a goal state G. That is,*

$$r_t = \begin{cases} 1 & \text{if } x_{t+1} = G \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Definition 7 (homogeneous problem solving task) *A task is a* homogeneous problem solving task *if it is a problem solving task and its associated state space is homogeneous with respect to the goal state G.*

Given the above definitions and descriptions we have the following result.

Theorem 8 *In a homogeneous problem solving task, the expected time needed by a* zero-initialized Q-learning *system to learn the actions along an optimal solution path is bounded below by the expression*

$$c_1 * \left\{ c_2 \left[\frac{P_+}{P_-}\right]^{k-i} \left[\left(\frac{P_+}{P_-}\right)^i - 1\right] + \frac{i}{1-2P_+} \right\} \quad (6)$$

where,

$$c_1 = \left(\frac{1}{1-P_=}\right),$$

$$c_2 = \frac{P_+}{(1-2P_+)^2},$$

$$P_= = \frac{b_=}{b_= + b_+ + b_-},$$

$$P_+ = \frac{b_+}{b_+ + b_-},$$

and

$$P_- = 1 - P_+,$$

and where i is the length of the optimal solution, and k is the depth bound on the state space (with respect to the goal).

Space does not permit a detailed derivation of Equation 6, however the key to the proof is to recognize that the time needed by a zero-initialized Q-learning system to first solve the task is exactly the time needed by a process to perform a random walk on a one dimensional state space which begins in state i and ends in state 0 — where the states are numbered left to right from 0 to $k-1$ and the walk takes a leftward step with probability $P_-(1-P_=)$, a rightward step with probability $P_+(1-P_=)$, and a step having no effect with probability $P_=$. That is, when solving the task for the first time the zero-initialized system performs an unbiased random walk over the state space. It chooses actions randomly until it stumbles into the goal state. By constraining our analysis to homogeneous problem solving tasks we are able to analyze this walk. In particular, it reduces to a random walk on a 1-dimensional state space. A detailed derivation is given in [Whitehead and Ballard, 1991].

Corollary 9 *For state spaces of* polynomial width *(see Definition 4), when $P_+ > 1/2$, the expected search time is moderately exponential in the state space size.*

In Equation 6, $P_=$ is the probability that the system, when choosing actions randomly, selects an action that leaves the distance to the goal unchanged, and P_+ (and P_-) is the conditional probability that the system chooses an action that increases (decreases) the distance to the goal given that it chooses one that changes the distance.

Figure 2 shows a series of plots of expected solution time (Equation 6) versus maximum distance k for a $i = 10$, and $P_+ \in [0.45, 0.55]$. When $P_+ > 1/2$, the solution time scales exponentially in k, where the base of the exponent is the ratio $\frac{P_+}{P_-}$. When $P_+ = 1/2$, the solution time scales linearly in k, and when $P_+ < 1/2$ it scales sublinearly.

The case where $P_+ > 1/2$ is important for two reasons. First, for many interesting problems it is likely that $P_+ > 1/2$. For example, if a robot attempts to build an engine by randomly fitting parts together, it is much more likely to take actions that are useless or move the system further from the goal than towards it. This follows since engine assembly follows a fairly sequential ordering. Similarly, a child can be expected to take time exponential (in the number of available building blocks) to build a specific object when combining them at random. Of course the state spaces for

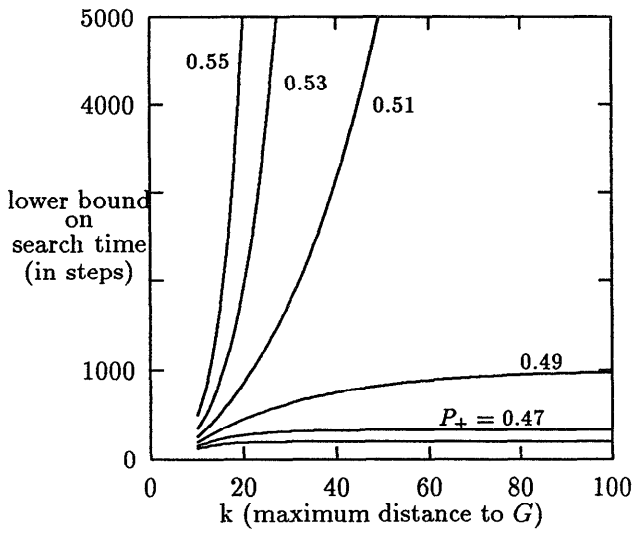

Figure 2: Search time complexity (lower bound) as a function of k.

building engines and assembling blocks are not homogeneous, but they may reasonably be approximated as such.

Second, when P_+ is only slightly greater than 1/2, it doesn't take long before the exponent leads to unacceptablely long searches. Figure 2 illustrates this point dramatically; even when P_+ is as small as 0.51 the solution time diverges quickly. When $P_+ = 0.55$ (i.e. the system is only 10% more likely to take a "bad" action than a "good" action), the search time diverges almost immediately.

Theorem 7 applies only to zero-initialized Q-learning systems. However, we expect these results to carry over to any learning system/algorithm that relies on unbiased search to initially solve tasks.

Cooperation for faster learning

Theorem 7 suggests that in the absence of a priori task knowledge, pure trial-and-error learning does not scale well with domain complexity (state-space size). Fortunately, a number of techniques can be used within the reinforcement learning paradigm to improve the learning rate.

One approach is avoid the problem altogether by providing the agent with an approximate controller *a priori*. This controller, by encoding *a priori* knowledge about the task, defines an initial policy that positively biases the search. In this case, trial-and-error experience is used primarily to compensate for modeling errors in the approximate controller [Franklin, 1988]. While this approach is useful for initial learning, its drawbacks are that it requires knowledge of the task *a priori* and it is less useful for adapting to change in the environment (i.e. low overall adaptability).

A second approach is to augment the agent with a predictive model and use it to perform hypothetical trial-and-error experiments [Sutton, 1990a; Sutton, 1990b; Whitehead and Ballard, 1989a; Whitehead and Ballard, 1989b]. This technique can improve the learning rate even when the predictive model itself is learned [Sutton, 1990a; Whitehead, 1989; Lin, 1990; Riolo, 1990]. The principle shortcoming of this approach is that it leads to agents who are only as good as their internal predictive models. That is, because multiple trials may be required before an accurate model can be learned, the model is useless during that first crucial trial when the agent first performs a task or first adapts to a change. Once, the agent has learned an accurate model (including learning the locations of rewards), hypothetical experiments begin to contribute.

A third approach, the focus of this paper, is to embed the agent in a cooperative social environment and develop algorithms for transferring knowledge. This approach is based on the idea that in nature, intelligent agents exist in a social structure that supports knowledge transfer between agents. Individuals learn from one another through social and cultural interactions: by watching each other, by imitating role-models, by receiving criticism from knowledgeable critics, and by receiving direction from supervisors. Knowledge transfer and trial-and-error learning interact synergistically. On the level of the individual, mechanisms for knowledge transfer dominate, and because of its inherent complexity, trial-and-error learning plays a lesser role — being used primarily to refine/elaborate knowledge already gained from others. On the level of the group, trial-and-error learning becomes an important tool for increasing the cumulative knowledge of the society as a whole. That is, even though agents, as individuals, cannot make discoveries at a useful rate, the inherent parallelism in the population can overcome the complexity of search and the group can accumulate knowledge and adapt at an acceptable rate.

The following sections describe and analyze two cooperative mechanisms for improving the adaptability of reinforcement learning systems. We call these *Learning with an External Critic* (LEC) and *Learning By Watching* (LBW) respectively.

LEC Analysis

The principle idea behind *learning with an external critic* is that the learner, while attempting to solve problems, is observed by a helpful critic, who analyzes the learners actions and provides immediate feedback on its performance. LEC algorithms achieve faster learning by reducing the delay between an action and its evaluation (i.e. feedback), mitigating the temporal credit assignment problem. LEC algorithms require only modest changes to the learner, since no interpretation is required. However, some interpretations

skills are required of the external critic. We have studied several LEC algorithms [Whitehead and Ballard, 1991]. In this paper we focus on one, called *Biasing Binary LEC.*

In the Biasing-Binary-LEC (or BB-LEC) algorithm, reward from the environment, r_w and reward from the external critic, r_c are treated separately. Reward from the environment is treated according to standard Q-learning (it is used to learn the action-value function), while the critic's reward is used to learn a biasing function, B, over state-action pairs. The biasing function is a simple weighted average of the immediate reward received from the critic. It is estimated using the rule:

$$B_{t+1}(x_t, a_t) \leftarrow (1 - \psi)B_t(x_t, a_t) + \psi r_c(t) \quad (7)$$

where $r_c(t)$ is the reward generated by the external critic in response to the agents action at time t and ψ is a decay factor between 0 and 1. We assume that at each time step, the critic generates a non-zero reward with probability p_{critic} according to the rule:

$$r_c(t) = \begin{cases} +R_c & \text{if } a_t \text{ is optimal} \\ -R_c & \text{otherwise} \end{cases} \quad (8)$$

where R_c is a positive constant.

The decaying average in Equation 7 is used to allow the agent to "forget" old advice that has not recently been repeated. Without it, the agent may have difficulty adapting to changes in the task once advice is extinguished.

The decision making rule for BB-LEC is simple. The agent sums the action-value and bias-value for each possible decision and chooses the decision with the largest total. That is,

$$\pi(x) = a \text{ such that } Q(x,a)+B(x,a) = \max_{b \in A}[Q(x,b)+B(x,b)].$$

Given the above LEC algorithm, we have the following weak upper bound on the search time.

Theorem 10 *The expected time needed by a zero-initialized BB-LEC system to learn the actions along an optimal path for a homogeneous problem solving task of depth k is bounded above by*

$$\frac{k}{P_{\text{critic}}} * |S| * b \quad (9)$$

where P_{critic} is the probability that on a given step the external critic provides feedback, $|S|$ is the total number of states in the state space, b is the branching factor (or total number of possible actions per state) and k is the depth of the state space.

This upper bound is somewhat disappointing because it is expressed in terms of the state space size, $|S|$, and the maximum depth, k. Our goal is to find algorithms that depend only upon task difficulty (i.e. length of optimal solution) and are independent of state space size and depth. Nevertheless the result is interesting for two reasons. First, it shows that when $\frac{P_+}{P_-} > 1/2$, BB-LEC is an improvement over pure zero-initialized Q-learning since the search time grows at most linearly in k whereas Q-learning grows at least exponentially in k. Second, because the upper bound is inversely proportional to P_{critic}, the theorem shows that even infrequent feedback from the critic is sufficient to achieve the linear upper bound. This has been observed in empirical studies, where even infrequent feedback from the critic substantially improves performance [Whitehead and Ballard, 1991].

The trouble with the LEC algorithm, as we've described it so far, is that the critic's feedback arrives late. That is, by the time the learner receives the critic's evaluation it finds itself in another (neighboring) state, where the feedback is of no value. If the learner had a means of returning to previously encountered states, it could make better use of the critic's feedback. This idea has lead to the following results, which show that under appropriate conditions the search time depends only upon the solution length and is independent of state space size.

Theorem 11 *If a zero-initialized Q-learning system using BB-LEC uses an inverse model to "undo" non-optimal actions (as detected based on feedback from the external critic) then the expected time needed to learn the actions along an optimal path for a homogeneous problem solving task is linear in the solution length i, independent of state space size, and is bounded above by the expression*

$$\left[\frac{2}{P_-(1 - P_=)} - 1\right] * i. \quad (10)$$

Similarly, if the task is structured so that the system can give up on problems after some time without success or if the system is continually presented with opportunities to solve new instances of a problem then previously encountered situations can be revisited without much delay and the search time can be reduced.

Theorem 12 *A zero-initialized Q-learning system using BB-LEC that quits a trial and starts anew if it fails to solve the task after n_q ($n_q \geq i$) steps has, for a homogeneous problem solving task, an expected solution time that is linear in i, independent of state space size, and is bounded from above by the expression*

$$\left(\frac{1}{P_-(1 - P_=)}\right) * n_q i. \quad (11)$$

Corollary 13 *A zero-initialized Q-learning system using BB-LEC that quits a trial and starts anew upon receiving negative feedback from the external critic has an expected solution time that is bounded from above by the expression*

$$\left(\frac{1}{P_+(1 - P_=)}\right) * \left(\frac{1}{P_= + (1 - P_=)P_+}\right) * i. \quad (12)$$

The crucial assumption underlying the above theorems is that the learner has some mechanism for quickly returning to the site of feedback, however for some tasks returning to a previous state may not be explicitly necessary to decouple search time from the state space size. In particular, if the optimal decision surface is smooth (i.e., optimal actions for neighboring states are similar), then action-value estimators that use approximators that locally interpolate (e.g. CMACs, or Neural Nets) can immediately use the critic's feedback to bias the impending decision. Similarly, if Q is approximated with overlapping hypercubes (e.g. classifier systems [Holland *et al.*, 1986]), then the critic's feedback can be expected to transfer to other situations as well. Although not reflected in the above theorems, we suspect that this observation is the basis of the ultimate power of LEC algorithms and will enable them to be useful even when explicit mechanisms for inversion are not available.

LBW Analysis

LEC algorithms are sensitive to naive critics. That is, if the critic provides poor feedback, the learner will bias its policy incorrectly. This limits the use of LEC algorithms to cases where the external critic is skilled and attentive. *Learning By Watching*, on the other hand, does not rely on a skilled, attentive critic. Instead, the learner gains additional experience by interpreting the behavior of others. If the observed behavior is skilled so much the better, but an LBW system can learn from naive behavior too.

In reinforcement learning, all adaptation is based on a sequence of state-action-reward triples that characterize the system's behavior. In LBW, the agent gets state-action-reward sequences not only by its own hand, but also by observing others perform similar tasks. In the analysis that follows, we assume that, at each time step, the learner can correctly recognize the state-action-reward triple of any agent it watches. This sequence is then used for learning just as if it were the learner's own personal experience. We also assume that the learner can observe the behavior of everyone in the population. Although these assumptions are overly simplistic and ignore many important issues, they are reasonable considering our goal — to illustrate the potential benefits of integrating reinforcement learning with cooperative mechanisms like "learning-by-watching."[3]

Given the above description of LBW, we can make the following observations.

Theorem 14 *The expected time required for a population of naive (zero-initialized) Q-learning agents using LBW to learn the actions along an optimal path decreases to the minimum required learning time at a rate that is $\Omega(1/n)$, where n is the size of the population.*

[3]Results similar to those described below can be obtained when the assumptions are relaxed.

Without help by other means, a population of naive LBW agent's may still require time exponential in the state space depth, however, search time can be decoupled from state space size by adding a knowledgeable role model.

Theorem 15 *If a naive agent using LBW and a skilled (optimal) role-model solve identical tasks in parallel and if the naive agent quits its current task after failing to solve it in n_q steps, then an upper bound on the time needed by the naive agent to first solve the task (and learn the actions along the optimal path) is given by*

$$\left\lceil \frac{i^2}{n_q} \right\rceil n_q + i. \tag{13}$$

As with LEC results, Theorem 14 relys on the agent having a mechanism for returning to previously encountered states. Intuitively this follows since when a naive agent and a skilled agent perform similar tasks in parallel, it is possible for the naive agent to move off the optimal solution path, and find itself in parts of the state space that are never visited by the skilled agent. Starting over is a means for efficiently returning to the optimal solution path. Again, we expect LBW systems to perform well on tasks that have decision surfaces that are smooth or can be represented by generalizing function approximators.

Conclusions

When used to solve multi-stage decision problems, reinforcement learning algorithms perform a kind of online, incremental search in order to find an optimal decision policy. The time complexity of this search strongly depends upon the size and structure of the state space and upon any knowledge encoded in the system *a priori*. When *a priori* knowledge is not available or when the system must adapt to a change in the environment, search can be excessive.

An analysis of the search time complexity for zero-initialized Q-learning systems indicates that for a restricted, but representative set of tasks, the search time scales at least exponentially in the depth of the state space. For polynomial width state spaces, this implies search times that are moderately exponential in state space size.

Learning with an External Critic (LEC) and *Learning By Watching* are two cooperative mechanisms that can substantially reduce search. LEC algorithms rely on feedback from an external critic to reduce feedback (reward) latency. LBW algorithms use observations of others as an auxiliary source of experience. Both algorithms reduce search and increase overall adaptability.

The LEC algorithm in its purest form, has a search time complexity that is at most linear in the state space size. Even though this is an improvement over pure Q-learning, the bound continues to depend on the state space size. The trouble with pure LEC is that, because the critic's evaluation is received after the fact, the

learner may find itself in another (neighboring) state, where the feedback has little value. When means exist for efficiently returning to states that have been previously visited (or states that are functionally equivalent) the search time can be decoupled from the state space size. This can be achieved either explicitly or implicitly. Explicit mechanisms include allowing the learner to use an inverse model and allowing the agent to restart (or pick a new instance of) a task if it fails to solve it after some time. Critic evaluation can immediately be made use of implicitly (or automatically) when the decision surface is smooth, so that neighboring state share the same optimal action; or when the policy can be represented by generalizing function approximators.

The advantage of LBW over LEC is that it doesn't necessarily rely on an attentive, knowledgeable critic. In particular, the search time complexity, of a population of naive LBW agents, scales as the inverse of the population size. When a knowledgeable (not necessarily attentive) role-model is available, a naive agent, under appropriate conditions, can learn the actions along the optimal solution path in time linear in the path length, (independent of state space size).

Although our results are for zero-initialized Q-learning systems solving homogeneous problem solving tasks, we expect them to apply equally to other reinforcement learning algorithms that depend on search. Our simulation studies [Whitehead and Ballard, 1991] support this hypothesis and also show LEC and LBW algorithms to be robust with respect to feedback noise and dropout.

Finally, it might be argued that what we're doing here is moving toward supervised learning, so why not abandon reinforcement learning altogether and focus on supervised learning. It is true that LEC and LBW algorithms take steps toward supervised learning by exploiting richer sources of feedback. Indeed, supervised learning (in its classic sense) could be incorporated into the LEC algorithm by expanding the vocabulary between the system and the external critic from its present "yes" or "no", to a list of possible actions. However, we want to retain reinforcement learning because it is such an autonomous (if weak) learning algorithm. Our philosophy is that an autonomous learning system should be able to exploit extra information available when learning (e.g., feedback from an outside critic), but it should not rely on it completely. A framework with reinforcement learning as its basic mechanism provides for such autonomy.

References

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuron-like elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[Franklin, 1988] Judy A. Franklin. Refinement of robot motor skills through reinforcement learning. In *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, TX, December 1988.

[Holland *et al.*, 1986] John H. Holland, Keith F. Holyoak, Richard E. Nisbett, and Paul R. Thagard. *Induction: processes of inference, learning, and discovery.* MIT Press, 1986.

[Jordan and Rumelhart, 1990] Michael I. Jordan and David E. Rumelhart. Supervised learning with a distal teacher. Technical report, MIT, 1990.

[Lin, 1990] Long-Ji Lin. Self-improving reactive agents: Case studies of reinforcement learning frameworks. In *Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, September 1990.

[Riolo, 1990] Rick L. Riolo. Lookahead planning and latent learning in classifier systems. In *Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, September 1990.

[Sutton, 1988] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Sutton, 1990a] Richard S. Sutton. First results with DYNA, an integrated architecture for learning, planning, and reacting. In *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[Sutton, 1990b] Richard S. Sutton. Integrating architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990. Morgan Kaufmann.

[Watkins, 1989] Chris Watkins. *Learning from delayed rewards.* PhD thesis, Cambridge University, 1989.

[Whitehead and Ballard, 1989a] Steven D. Whitehead and Dana H. Ballard. Reactive behavior, learning, and anticipation. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, 1989.

[Whitehead and Ballard, 1989b] Steven D. Whitehead and Dana H. Ballard. A role for anticipation in reactive systems that learn. In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989. Morgan Kaufmann.

[Whitehead and Ballard, 1991] Steven D. Whitehead and Dana H. Ballard. A study of cooperative mechanisms for faster reinforcement learning. TR 365, Computer Science Dept., University of Rochester, Feburary 1991. (A shorter version to appear AAAI-91).

[Whitehead, 1989] Steven D. Whitehead. Scaling in reinforcement learning. Technical Report TR 304, Computer Science Dept., University of Rochester, 1989.