Reinforcement Learning with Time

Daishi Harada
daishi@cs.berkeley.edu
Dept. EECS, Computer Science Division
University of California, Berkeley

Abstract

This paper steps back from the standard infinite horizon formulation of reinforcement learning problems to consider the simpler case of finite horizon problems. Although finite horizon problems may be solved using infinite horizon learning algorithms by recasting the problem as an infinite horizon problem over a state space extended to include time, we show that such an application of infinite horizon learning algorithms does not make use of what is known about the environment structure, and is therefore inefficient. Preserving a notion of time within the environment allows us to consider extending the environment model to include, for example, random action duration. Such extentions allow us to model non-Markov environments which can be learned using reinforcement learning algorithms.

Introduction

This paper reexamines the notion of time within the framework of reinforcement learning. Currently, reinforcement learning research focuses on the infinite horizon formulation of its problems; finite horizon problems may occasionally be mentioned, but are rarely studied explicitly (Kaelbling, Littman, & Moore 1996). Within the infinite horizon formulation, the goal is to find the optimal stationary policy. Stationary policies, however, are exactly that: stationary. They do not allow time-dependencies in the policies they represent; indeed, the notion of time is essentially finessed away. This paper focuses on developing reinforcement learning algorithms to solve finite horizon problems.

The key idea underlying reinforcement learning algorithms is to combine techniques from the theories of stochastic approximation and dynamic programming. This allows reinforcement learning algorithms to solve problems which are of the same form as dynamic programming problems, but contain unknown parameters. A finite horizon problem, by its nature, distinguishes between different points in time. There is an explicit "end" to which one can be near or far. When the model parameters are fully known, the classic dynamic programming solution to such a problem is a policy which varies with each time step. The question, then, is how to find such a policy when the model parameters are unknown. Of course, any finite horizon problem may be restated as an infinite horizon problem over an extended state space which includes time; in other words, any finite horizon problem may be solved using an algorithm which solves infinite horizon problems. However, this is generally not very efficient, since an algorithm intended for an infinite horizon problem does not fully utilize the known structure of a finite horizon problem environment.

By explicitly addressing finite horizon problems, it is possible to design reinforcement learning algorithms which are much more efficient; such an algorithm is presented in this paper. The basic idea behind the algorithm is that each experience provides us with information relevant to more than a single value parameter. This allows us to update many parameters per experience, as opposed to standard algorithms which update a single parameter. In addition, there are other advantages to explicitly considering finite horizon problems. For example, we can model properties such as action duration. In general this allows the modelled environment to be non-Markov. However, since the extended state-time environment retains the Markov property, it is clear that such environments may be learned; what is not as clear is whether it is possible to still take advantage of the environment structure to learn efficiently. We show that this is in fact the case, and present such an algorithm.

A risk vs. no-risk game

Let us clarify the ideas introduced above by first considering the following simple game. In this game, the player has two moves: a "risky" move and a "no-risk" move. The risky move wins with probability p and loses otherwise, while the no-risk move wins with probability q but otherwise allows the player to "try again," and hence never loses (Figure 1.)

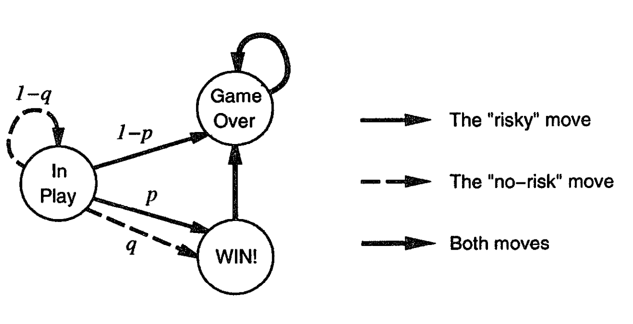

Figure 1: A simple risk vs. no-risk game. InPlay is the start state with reward 0. The states WIN! and GameOver have rewards 1 and 0, respectively.

The standard Q-learning/infinite-horizon dynamic programming framework provides us with a technique for finding the optimal stationary policy g^* for this game. Here optimal is defined to mean the policy which maximizes the cost function

$$E\left[\sum_{t=0}^{\infty}\gamma^t R(X_t^g)\right] = \sum_{t=0}^{\infty}\gamma^t E[R(X_t^g)]$$

where γ is the discount factor, X_t^g is a random variable representing the state at time t under policy g, and $R(X_t^g)$ is a random variable representing the reward associated with state X_t^g.

In the case of this particular game, we find the optimal stationary policy to be

$$g^*(\mathsf{InPlay}) = \begin{cases} \text{risky move} & \text{if } p \geq \frac{q}{1-\gamma+\gamma q} \\ \text{no-risk move} & \text{otherwise} \end{cases}$$

Note that the other two states, WIN! and GameOver, provide no choices to the player and hence are not relevant towards specifying a policy. When $\gamma = 0$, the condition for selecting the risky move reduces to $p > q$; in other words only one step into the future is relevant to the decision. On the other hand, when $\gamma = 1$, the condition reduces to $p \geq 1$. Since p is a probability, this condition can only be satisfied by $p = 1$; if the risky move is a sure win, then there is no reason to consider the no-risk move. For $p < 1$, the optimal policy is to never take the risky move. This is natural, since equally weighting all times in the future is equivalent to saying that the objective is to win eventually, and hence there is no reason to make the risky move.

Let us now consider the finite-horizon case, where the player has a time-limit/horizon T. Assuming that time is discrete and starts at 0, we find that the optimal (non-stationary) policy for $0 \leq t < T$ is

$$g^*(\mathsf{InPlay}, t) = \begin{cases} \text{risky move} & \text{if } p > q \text{ and} \\ & \quad t = T-1 \\ \text{no-risk move} & \text{otherwise} \end{cases}$$

This policy simply states that if $p \leq q$, then never make the risky move; otherwise, if $p > q$, only make the risky move at the last possible moment, when $t = T - 1$.

Although simple, this example shows how non-stationary policies can express more sophisticated control strategies than stationary ones. Of course, there is a cost associated with this increase in expressiveness: the policy is now also a function of time, and hence the encoding of the policy will be correspondingly larger.

The addition of random action duration makes even a simple environment such as this risk vs. no-risk game interesting. Consider letting the two actions risk and no-risk have durations following the distributions $\pi_r(t)$ and $\pi_{nr}(t)$, respectively, as given below:

	Duration t =		
	1	2	3
$\pi_r(t)$ =	0.3	0.7	0.0
$\pi_{nr}(t)$ =	0.6	0.1	0.3

(In this example, the action durations are bounded above, by 2 and 3.) What this means, for example, is that if the risky move is taken in state InPlay at time t, the probability of ending in state WIN! at time $t+2$ is $\pi_r(2) \times p = 0.7p$. Note that although we explicitly model time as a distinguished variable t, our transition probabilities are defined over the extended state space (s, t), and hence the model makes no claims as to the state of the game at the intermediate time $t+1$. If, in addition, we fix $p = 0.8$, $q = 0.5$, and let the horizon $T = 5$, then the optimal policy and corresponding value function for InPlay may be easily computed using dynamic programming. In particular, we find them to be

Time	Optimal Move	Value
0	no-risk move	0.9075
1	no-risk move	0.825
2	risky move	0.8
3	risky move	0.8
4	no-risk move	0.3
5	—horizon/end game—	

We see that the optimal end-game strategy is no longer as simple as in the previous case. In particular, it is no longer optimal to make the risky move at the last (decision) step, since the risky move takes more than 1 time step with relatively high probability. Note also that in general, adding action duration to the environment model makes the environment non-Markov, since the state at time $t+1$ is not conditionally indepedent of the state history given the state at time t (it is only possible to add a finite number of auxiliary "in-transit" states to create a Markov model of an environment with action duration if the maximum duration is bounded above or if the duration is memoryless.)

We now consider how to learn such non-stationary policies given that the model parameters are unknown. We first consider the case without action duration.

Finite-horizon Q-learning

Let us define the problem to be addressed and fix our notation. Let $\mathcal{X}$ be a finite set of states, $\mathcal{U}$ be a finite set of actions, and $\mathcal{T} = \{0, 1, \cdots, T\}$ be a finite set of points in time. Let $P_{xy}(u)$, where $x, y \in \mathcal{X}, u \in \mathcal{U}$, denote the transition probability of going from x to y under u, and let $R(x)$, where $x \in \mathcal{X}$, denote the random variable with finite mean $\bar{R}(x)$ representing the reward associated with state x. A policy g is a mapping from $\mathcal{X} \times \mathcal{T}$ to $\mathcal{U}$. Given a policy g and a probability density function over $\mathcal{X}$ for the initial state at time 0, the random variables $\{X_t^g \mid t \in \mathcal{T}\}$ representing the state at time t while acting according to g are well defined, and hence we may define the expected cost under this policy to be $J(g) = E[\sum_{t \in \mathcal{T}} R(X_t^g)]$. The problem is to find the optimal policy g^*, where $g^* = \text{argmax}_g J(g)$.

When $\{P_{xy}(u)\}$ and $\{\bar{R}(x)\}$ are known, this is the standard finite-horizon Markov decision process problem, and hence may be solved using classic dynamic programming (Bertsekas 1987). The goal of this section is to develop a learning algorithm for determining the optimal policy when $\{P_{xy}(u)\}$ and $\{\bar{R}(x)\}$ are unknown. Of course, since $|\mathcal{X}|$ and $|\mathcal{T}|$ are both finite, standard Q-learning on the extended state space $\mathcal{X} \times \mathcal{T}$ may be used to solve this problem. Let us first consider this standard solution, and briefly review the Q-learning framework.

Consider the following function over the extended state space $\mathcal{X} \times \mathcal{T}$:

$$V(x,t) = E\left[\sum_{\tau \geq t} R(X_\tau^{g^*})\right] \tag{1}$$

where $x \in \mathcal{X}, t \in \mathcal{T}$, and $X_t^{g^*} = x$. Intuitively, $V(x,t)$ is the expected cost-to-go given that one is in state x at time t, and acts according to the optimal policy. Given $V(x,t)$ it is straightforward to obtain the optimal policy: Let $S(x,u)$ be the random variable over $\mathcal{X}$ which is the succesor state of x having done u. Then the optimal action in extended state (x,t) is $\text{argmax}_u E[V(S(x,u), t+1)]$. Hence we may rewrite Equation (1) as

$$\begin{aligned} V(x,t) &= \bar{R}(x) + \max_u E[V(S(x,u), t+1)] \\ &= \max_u (E[R(x) + V(S(x,u), t+1)]) \end{aligned}$$

This may be further rewritting by introducing what is called the Q-function:

$$\begin{aligned} V(x,t) &= \max_u Q((x,t),u) \\ Q((x,t),u) &= \bar{R}(x) + E[V(S(x,u), t+1)] \end{aligned}$$

The goal of Q-learning is to find such a function Q that satisfies this equilibrium constraint. Since $\{P_{xy}(u)\}$ and $\{\bar{R}(x)\}$ are unknown, the Q-function must be estimated. In particular, Q-learning uses the following update rule/estimator. Given the experience $(x,t) \xrightarrow{u} (y, t+1)$ with reward r for state x:

$$\begin{aligned} Q_{n+1}((x,t),u) \leftarrow\ & (1-\alpha_n)Q_n((x,t),u) + \\ & \alpha_n(r + V_{n+1}(y,t+1)) \end{aligned} \tag{2}$$

where

$$V_{n+1}(y,t+1) \leftarrow \max_v Q_n((y,t+1),v) \tag{3}$$

whenever $t < T-1$, and if $t = T-1$

$$V_{n+1}(y,t+1) = (1-\alpha_n)V_{n+1}(y,t+1) + \alpha_n r' \tag{4}$$

where r' is the reward for state y. Q_n and V_n represent the estimate of Q and V after the nth observation.

This update rule, however, does not make use of the known structure of the underlying problem, and is therefore inefficient with respect to the number of experience samples. Since the unknown parameters $\{P_{xy}(u)\}$ and $\{\bar{R}(x)\}$ do not depend on time, each experience $(x,t) \xrightarrow{u} (y,t+1)$ with reward r provides us with information that can be used to update $Q((x,\tau),u)$ for all $0 \leq \tau < T$. More precisely, each such experience provides us with the ability to compute an unbiased sample of $R(x) + V(S(x,u), \tau+1)$ for all $0 \leq \tau < T$. Hence, since the equilibrium constraints are still the same, we may simply apply Equation (2) for all τ to obtain the following update:

$$\begin{aligned} Q_{n+1}((x,\tau),u) \leftarrow\ & (1-\alpha_n)Q_n((x,\tau),u) + \\ & \alpha_n(r + V_{n+1}(y,\tau+1)) \end{aligned}$$

where $V_{n+1}(y,\tau+1)$ is again defined by Equations (3) and (4). Note that with α_n chosen appropriately, this update rule satisfies the conditions of the convergence Theorem in (Jaakola, Jordan, & Singh 1994), and hence converges with probability 1.

Simulation results

Let us compare this new algorithm, which we call Q_T-learning, to the standard one using the simple risk vs. no-risk game environment without action duration described above. The parameters of the environment were set as follows: $T = 5$, $p = 0.8$, and $q = 0.5$.

Both learning algorithms were configured in the following way. The inital estimates of the Q-values were chosen uniformly at random in the range $[0.0, 0.1]$. The exploration policy was to select, with probability p_e, an action uniformly at random; with probability $1 - p_e$ the algorithm would select what it currently (i.e., for the current values of n, x, and t) considered the best action: $\text{argmax}_u Q_n((x,t),u)$. The exploration probability p_e was varied over the values $\{0.05, 0.1, 0.15, 0.2, 0.25\}$. The learning rate α was held constant for each run, and was varied over the values $\{0.005, 0.01, 0.025, 0.05, 0.1\}$ across runs. For each combination of the parameters there were 4 runs, for a total of 100 runs for each algorithm.

Two performance measures were used to evaluate the learning behavior of the algorithms. The first was the sum of squared error of the estimated value function for the state InPlay for $0 \leq i < T = 5$. The second was the policy error for the same (state, time) pairs. In standard Q-learning, for each experience sample exactly one parameter is updated. Hence measuring the

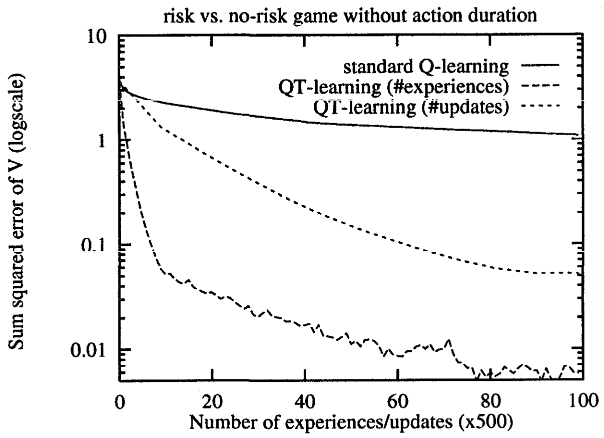

Figure 2: Comparison of the sum of squared error of the value function between standard Q-learning and Q_T-learning on the risk vs. no-risk game *without* action duration. For Q_T-learning, the performance is shown both against the number of experience samples seen by the algorithm (#experiences) and the number of parameters updated (#updates).

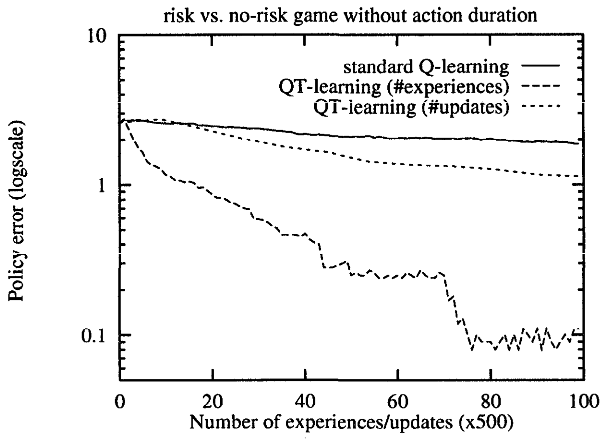

Figure 3: Comparison of the policy error between standard Q-learning and Q_T-learning on the risk vs. no-risk game *without* action duration. For Q_T-learning, the performance is shown both against the number of experience samples seen by the algorithm (#experiences) and the number of parameters updated (#updates).

performance against the number of experience samples seen by the Q-learning algorithm is equivalent to measuring the performance against the number of parameters updated. However, this does not hold for Q_T-learning, since many parameters are updated per experience sample. The performance of the Q_T-learning algorithm is therefore shown against both.

The resulting performance curves, which are the averages of the 100 runs, are shown in Figures 2 and 3. The results are somewhat surprising. Even when the performance of Q_T-learning is measured against the number of parameters updated, Q_T-learning outperforms standard Q-learning. It was expected that since both Q_T-learning and Q-learning use the same update function with respect to any single parameter, their performance when compared against the number of updates would be essentially the same. Indeed, the preliminary data that was collected for this simulation experiment had coincided with this intuition; the more complete results shown here seem to indicate that Q_T-learning performs even better than one might have expected.

What is unambiguously clear, however, is that Q_T-learning is significantly more data/experience efficient. Intuitively, this efficiency follows from the number of degrees of freedom in the parameters assumed by each algorithm. Although both algorithms estimate $|\mathcal{X}||\mathcal{T}||\mathcal{U}|$ parameters, the Q-learning algorithm estimates the parameters assuming that each is a distinct degree of freedom. The Q_T-learning algorithm, on the other hand, does not assume this; it makes use of the fact that the parameters are in fact not unrelated, and that $|\mathcal{X}|^2|\mathcal{U}| + |\mathcal{X}|$ parameters suffice to fully specify the environment model. Now $\{P_{xy}(u)\}$ and $\{\bar{R}(x)\}$, the actual parameters which specify the environment, do not depend on time. Therefore the convergence rate of the performance measure also does not depend on $|\mathcal{T}|$. Of course, this dependence cannot be entirely eliminated since the size of the policy is $O(|\mathcal{T}|)$. However, the dependence is contained entirely within the learning procedure, which updates $O(|\mathcal{T}|)$ parameters per data sample.

Actions with random duration

We now consider extending the model to incorporate random action duration. Let $\pi_u(t)$ be a probability density function over $\{1, 2, \cdots\}$ representing this duration for action u. Then we may define the following transition probabilities for all $x, y \in \mathcal{X}$, $u \in \mathcal{U}$, and $t_1, t_2 \in \mathcal{T}$:

$$P'_{(x,t_1)(y,t_2)}(u) = P_{xy}(u)\pi_u(t_2 - t_1) \quad (5)$$

This may be viewed as saying that

$$\begin{aligned} & \Pr\{Y, T_2 \mid X, U, T_1\} \\ = & \Pr\{Y \mid X, U, T_1\} \Pr\{T_2 \mid Y, X, U, T_1\} \\ = & \Pr\{Y \mid X, U\} \Pr\{T_2 - T_1 \mid U\} \end{aligned}$$

The important point to note is that this transition probability is stationary; i.e., only the difference $t_2 - t_1$ is relevant. The model described previously is the special case when $\pi_u(t) = 1$ if $t = 1$ and is 0 otherwise. Again, it is clear that the problem of finding an optimal policy in this environment may be solved by using standard techniques on an extended state space. However, as before, we would like to make use of our prior knowledge concerning the structure of the environment to develop a more efficient algorithm. Let us first consider the equilibrium relation which must be satisfied by the optimal Q-values:

$$\begin{aligned} & Q^*((x,t_1),u) \\ = \; & \bar{R}(x) + \sum_{y\in\mathcal{X},t_2\in\mathcal{T}} P'_{(x,t_1)(y,t_2)}(u)V^*(y,t_2) \\ = \; & \bar{R}(x) + \sum_{y\in\mathcal{X},t_2\in\mathcal{T}} P_{xy}(u)\pi_u(t_2-t_1)V^*(y,t_2) \end{aligned}$$

where the definition of $V^*(y,t_2)$ is analogous to that of $V(y,t)$ in Equations (3) and (4). We see that this extension to include uncertain action duration has not significantly increased the complexity of the estimation problem.

There is, however, one issue which must be addressed. Namely, it is necessary to more carefully specify what we now mean by the time-limit/horizon T. Let us distinguish between "hard" and "soft" time-limits. A hard time-limit treats all times after T as if they do not exist. A soft time-limit, on the other hand, simply ignores rewards obtained after time T; it does not "stop time" at T. When the time-limit is hard, the environment is no longer stationary, and requires that we add an auxiliary final (time-limit exceeded) state F. Equation (5) above is correspondingly modified to become

$$P'_{(x,t_1)(y,t_2)}(u) = \begin{cases} P_{xy}(u)\pi_u(t_2-t_1) & \text{if } t_2 \le T \\ 0 & \text{if } t_2 > T \end{cases}$$

for $y \neq F$, and for $y = F$:

$$P'_{(x,t_1)(F,t_2)}(u) = \begin{cases} P_{xy}(u)\sum_{\tau\ge T}\pi_u(\tau-t_1) & \text{if } t_2 = T \\ 0 & \text{if } t_2 \neq T \end{cases}$$

The importance of this distinction is clear. If the enforcement of the time-limit is soft, then it is possible to have experiences $(x,t_1) \xrightarrow{u} (y,t_2)$ where $t_2 > T$; this allows us to construct from the experience the appropriate unbiased sample for all $0 \le \tau < T$ as before. Hence the update rule given an experience $(x,t_1) \xrightarrow{u} (y,t_2)$ with reward r is for all $0 \le \tau < T$

$$\begin{aligned} Q_{n+1}((x,\tau),u) \;\leftarrow\; & (1-\alpha_n)Q_n((x,\tau),u) + \\ & \alpha_n(r + V_{n+1}(y,\tau+(t_2-t_1))) \end{aligned}$$

where the definition of $V_{n+1}(y,t')$ is extended so that $V_{n+1}(y,t') = 0$ for all $t' \ge T$. It is natural that this algorithm, as in the simpler case, would have the performance advantages over a standard Q-learning algorithm running over the $\mathcal{X} \times \mathcal{T}$ state space.

On the other hand, if the enforcement of the time-limit is hard, the extention is not as straightforward. Experiences where $t_2 = T$ must certainly be treated separately, and would allow us to update only a single parameter. In addition, even when $t_2 < T$, only those parameters for τ such that $\tau + (t_2 - t_1) < T$ may be updated. Hence the updating procedure on experience $(x,t_1) \xrightarrow{u} (y,t_2)$ with reward r becomes:

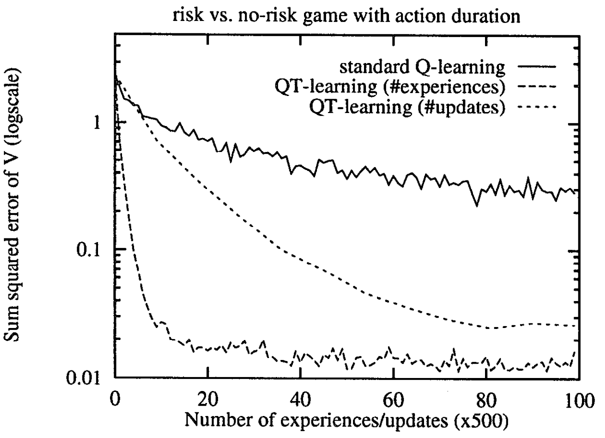

Figure 4: Comparison of the sum squared error of the value function between standard Q-learning and Q_T-learning on the risk vs. no-risk game *with* action duration. For Q_T-learning, the performance is shown both against the number of experience samples seen by the algorithm (#experiences) and the number of parameters updated (#updates).

If $t_2 < T$ then for all τ such that $0 \le \tau < T - (t_2 - t_1)$

$$\begin{aligned} Q_{n+1}((x,\tau),u) \;\leftarrow\; & (1-\alpha_n)Q_n((x,\tau),u) + \\ & \alpha_n(r + V_{n+1}(y,\tau+(t_2-t_1))) \end{aligned}$$

Otherwise, if $t_2 = T$, then

$$\begin{aligned} Q_{n+1}((x,t_1),u) \;\leftarrow\; & (1-\alpha_n)Q_n((x,t_1),u) + \\ & \alpha_n(r + V_{n+1}(y,t_2)) \end{aligned}$$

The efficiency of this algorithm when compared to standard Q-learning would depend on the difference between the mean duration of actions and T. If T is sufficiently large compared to the mean action durations, then this algorithm will have performance advantages similar to those presented before.

Simulation results

This last algorithm for environments with hard time-limits was tested using the risk vs. no-risk game with action duration described at the beginning of this paper. The learning algorithm was configured in the same manner as in the previous simulation experiment. The results are shown in Figures 4 and 5.

Although the Q_T-learning algorithm still shows significant efficiency over the standard algorithm, it would appear that this more sophisticated environment is correspondingly more difficult to learn. On the other hand, the standard Q-learning algorithm appears to perform better. This is not unexpected, since the addition of action duration connects the extended state space more densely, and hence allows information concerning updated parameter values to filter more readily to the rest of the estimates.

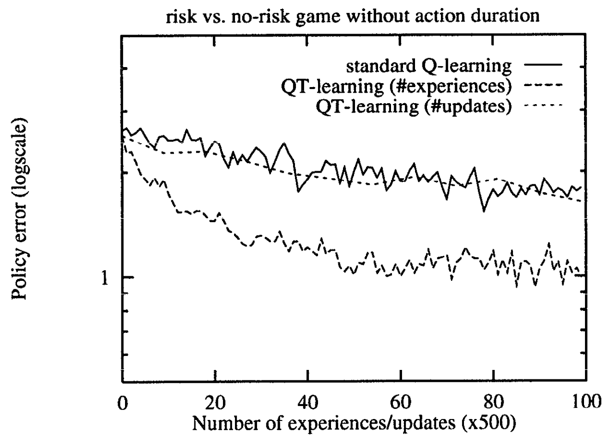

Figure 5: Comparison of the policy error between standard Q-learning and Q_T-learning on the risk vs. no-risk game *with* action duration. For Q_T-learning, the performance is shown both against the number of experience samples seen by the algorithm (#experiences) and the number of parameters updated (#updates).

Discussion and future work

There are several notable disadvantages of the finite horizon formulation as presented above. First, it requires that the horizon T be known. Second, the $O(|T|)$ size required to represent the policies may make the algorithms impractical. In this final section we consider ways in which these problems might be addressed.

First let us consider relaxing the condition that T be known. In particular, consider reformulating the problem so that T be a random variable with values $\{0, 1, \cdots\}$ representing the remaining lifetime of the agent, and suppose that the current state X_0 is fixed and known. Then the cost function given a policy g is

$$\begin{aligned} & E\left[\sum_{t=0}^{T} R(X_t^g)\right] \\ = & \sum_{\tau=0}^{\infty} \Pr\{T=\tau\}\, E\left[\sum_{t=0}^{T} R(X_t^g) \,\middle|\, T=\tau\right] \\ = & \sum_{t=0}^{\infty} E[\,R(X_t^g)\,] \sum_{\tau=t}^{\infty} \Pr\{T=\tau\} \\ = & \sum_{t=0}^{\infty} E[\,R(X_t^g)\,] \Pr\{T \geq t\} \end{aligned}$$

where we have assumed that the rewards $R(X_t^g)$ are independent of T. It follows that when the distribution of T is geometric with parameter γ,

$$E\left[\sum_{i=0}^{T} R(X_t^g)\right] = \sum_{i=0}^{\infty} \gamma^t E[\,R(X_t^g\,]$$

which is the standard discounted cost function for infinite horizon problems. Alternatively, discounting may be viewed as equivalent to extending the state space with a terminal, 0-reward state to which all actions lead with probability $1-\gamma$, and using the average cost function. In either case, we obtain the standard infinite horizon model because when the distribution of T is geometric, the optimal policy is stationary. It is also clear that the geometric distribution is the only distribution of T for which the optimal policy is stationary.

The second disadvantage, that of representation size, is further aggravated when T is unknown: in general, for arbitrary distributions of T, non-stationary policies would require an infinite-sized table to represent. It would seem that the goal must be weakened to that of finding an approximation to the optimal policy. An approach towards this is to define an appropriate family of policies and to search only for the optimal policy within this family. In particular, we would like to consider policies which fit the following form:

$$Q((s,t),u) \;=\; f_\theta\,(Q(s,u), t, [s,u])$$

for some f_θ, where $Q(s,u)$ is the Q-value given by the standard infinite-horizon algorithm, and $[s,u]$ is used to indicate that these values (s and u) are optional. This would allow us to extend existing algorithms to generate non-stationary policies.

Although good candidates for f_θ have yet to be found, given such, interesting questions to be asked include:

- Is it possible to characterize the competitive ratio (c.f. (Ben-David *et al.* 1990)) between the best policy in a policy family and the optimal policy?
- What is the trade-off between the complexity of a policy family and the loss incurred by the best policy in the family over the optimal one?

References

Ben-David, S.; Borodin, A.; Karp, R.; Tardos, G.; and Wigderson, A. 1990. On the power of randomization in online algorithms. In *Proceedings of the* 22nd *Annual ACM Symposium on Theory of Computing*, 379–386. Baltimore, MD: ACM.

Bertsekas, D. 1987. *Dynamic Programming: Deterministic and Stochastic Models.* Englewood Cliffs, N.J. 07632: Prentice-Hall, Inc.

Jaakola, T.; Jordan, M.; and Singh, S. 1994. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation* 6:1185–1201.

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