Learning to Achieve Goals

Leslie Pack Kaelbling[*]
Computer Science Department
Box 1910
Brown University
Providence, RI 02912 USA
lpk@cs.brown.edu

Abstract

Temporal difference methods solve the temporal credit assignment problem for reinforcement learning. An important subproblem of general reinforcement learning is learning to achieve dynamic goals. Although existing temporal difference methods, such as Q learning, can be applied to this problem, they do not take advantage of its special structure. This paper presents the DG-learning algorithm, which learns efficiently to achieve dynamically changing goals and exhibits good knowledge transfer between goals. In addition, this paper shows how traditional relaxation techniques can be applied to the problem. Finally, experimental results are given that demonstrate the superiority of DG learning over Q learning in a moderately large, synthetic, non-deterministic domain.

1 Introduction

Reinforcement learning is a general tool for deriving strategies that optimize a fixed reinforcement function in a probabilistic environment. A crucial problem in reinforcement learning is *temporal credit assignment*: how to choose actions based on good results that happen after (perhaps long after) the action is taken. This problem is solved well in the general case by *temporal difference* methods, such as Watkins' Q learning [Barto *et al.*, 1989; Watkins, 1989] and Sutton's TD algorithm [Sutton, 1988].

In much of the work on reinforcement learning, however, researchers have studied a restriction of the problem to cases of "goals of achievement." Rather than having an arbitrary mapping of states of the world to reinforcements for the agent, there is a single "goal" state at which the agent must arrive as soon as possible. Goals of achievement can be modeled in the general reinforcement-learning framework by having the goal state generate a positive reinforcement and all other states a zero reinforcement. Because there is a decay factor built into the temporal-difference methods, actions that lead to the goal sooner rather than later will be preferred.

Domains with goals of achievement can be learned by methods that are tailored to this special case. In this paper, we present the *DG learning* algorithm, which is analogous to Q learning, but directly suited to goals of achievement. In the basic single-goal case, DG learning is somewhat more effective than Q learning.

The real importance of DG learning is shown when we consider the case of goals of achievement that change over time; the common scenario of a taskable robot is an instance of this sort of problem. A simple extension to DG learning will allow a large amount of between-task transfer, making it greatly preferable to Q learning for this sort of problem, without requiring any complex mechanisms.

Finally, we consider the ability to learn from interaction with a model, as is done in Sutton's Dyna system. We will see that such learning can be very naturally integrated into the DG-learning framework.

2 Q Learning

We assume that a learning agent is embedded in an environment in such a way that it can discriminate the set $\mathcal{S}$ of distinct world situations and can take the set $\mathcal{A}$ of actions on the world. The world is modeled as a Markov process, making stochastic transitions based on its current state and the actions taken by the agent. We let $T(s, a, s')$ be the probability that the world will transition to state s' given that it was in state s and the agent executed action a. In addition, for each state s and action a, $r(s, a)$ is the *reinforcement value* of taking action a in situation s. In general, this value is a scalar random variable; it must have a stationary distribution, but the same situation-action pair may have different results on different trials.

The general reinforcement-learning problem is typically stated as finding a policy that maximizes expected discounted reinforcement. A policy π is mapping from $\mathcal{S}$ to $\mathcal{A}$. The expected discounted reinforcement of a policy π in a situation s is defined as

$$\sum_{t=0}^{\infty} \gamma^t er(t) \ ,$$

[*]This work was supported in part by a National Science Foundation National Young Investigator Award IRI-9257592 and in part by ONR Contract N00014-91-4052, ARPA Order 8225.

where $er(t)$ is the expected value of the reinforcement obtained at step t given that the agent started in situation s and executed policy π. The variable γ is the *discounting factor*; it controls to what degree rewards in the distant future affect the total value of a policy and is usually just slightly less than 1.

Given definitions of the transition probabilities and the expected reinforcements, it is possible to solve for the optimal policy, using methods from dynamic programming [Bellman, 1957; Howard, 1960]. A more interesting case occurs when we wish to simultaneously learn the dynamics of the world and construct the policy. Watkins' Q learning algorithm gives us an elegant and efficient method for doing this.

Let $Q^*(s,a)$ be the expected discounted reinforcement for taking action a in situation s and continuing thereafter with the optimal policy. It can be recursively defined as

$$Q^*(s,a) = er(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a' \in \mathcal{A}} Q^*(s',a') \ .$$

Because we do not know T and er initially, we construct incremental estimates of the Q values on line. Starting with $Q(s,a)$ at any value (but typically 0), every time an action is taken, update the Q values as follows:

$$Q(s,a) := (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a' \in \mathcal{A}} Q(s',a')) \ ,$$

where r is the actual reinforcement value received for taking action a in situation s, s' is the next state, and α is a learning rate (between 0 and 1).

Given the Q values, there is a policy defined by taking, in any situation s, the action a that maximizes $Q(s,a)$. Watkins showed [Watkins, 1989] that, given some assumptions, including that every state-action pair is tried infinitely often on an infinite run, the Q values will converge to the true Q^* values, and hence the induced policy will converge to the optimal one.

Of course, in any practical situation, as the learned Q values converge to the true Q^* values, we will have to use the policy to control action. But in doing so, we are in danger of violating the requirement that every state-action pair be tried infinitely often. This is an instance of the exploration versus exploitation trade-off; it has been treated extensively elsewhere [Kaelbling, 1993b; Thrun, 1992]. In the interest of simplicity in this paper, we generate actions probabilistically based on the Q values using a Boltzmann distribution. Given a situation s, we choose action a with probability

$$\frac{e^{Q(a,s)/T}}{\sum_{a \in \mathcal{A}} e^{Q(a,s)/T}} \ .$$

This serves to make actions whose values are much better than the others be chosen with much greater likelihood. The *temperature* parameter T controls the amount of exploration (the degree to which actions other than the one with the best Q value are taken).

3 DG Learning

The DG learning (DG stands for *dynamic goal*) method applies only to domains in which the aim of the agent is to arrive at some goal state in the smallest number of steps. We still assume that the world makes stochastic transitions, but the goal is explicitly named and there is no reinforcement function. The aim of DG learning, at its simplest, is to arrive at the policy that minimizes the expected number of steps to the goal. The method is applicable without change to the case where the goal varies on line: now a policy is a mapping from $\mathcal{S} \times \mathcal{S}$ to $\mathcal{A}$, specifying an action to take based on the current situation and the goal situation.

We define $DG^*(s,a,g)$ to be the expected number of steps required to get to situation g from situation s by starting with action a and continuing with the optimal policy. It is conveniently described recursively as

$$DG^*(s,a,g) = 0$$

if $s = g$ and

$$DG^*(s,a,g) = 1 + \sum_{s' \in \mathcal{S}} T(s,a,s') \min_{a' \in \mathcal{A}} DG^*(s',a',g)$$

otherwise.

Rather than learning T then calculating DG, we estimate DG directly on line. Starting with $DG(s,a,g)$ at any value (but typically 0), every time an action is taken, update the DG values as follows:

$$DG(s,a,g) := (1-\alpha)DG(s,a,g) + \alpha(1 + \min_{a' \in A} DG(s',a',g)),$$

where α is the learning rate and s' is the next state.

A policy can be directly constructed from the DG values by choosing the action a that minimizes $DG(s,a,g)$ for every current situation s and goal situation g. Although no theoretical results have yet been developed with respect to this learning procedure, we conjecture that it can be shown to converge under restrictions similar to those for the Q-learning result.

The trade-off between exploration and exploitation is important in DG learning as well; we use the Boltzmann distribution to construct action probabilities, though interval estimation techniques [Kaelbling, 1993b] could be employed for more careful exploration.

4 Learning From a Model

In on-line learning tasks, every step taken by the learning algorithm is a step taken by the agent in the world. In many cases, the agent is incurring risk by taking exploratory actions and we would like to minimize such risk. One way to decrease the risk is to employ internal learning from a model rather than direct learning from the world.

4.1 Dyna and Q Learning

Sutton [Sutton, 1990] and Whitehead and Ballard [Whitehead and Ballard, 1989] have both explored the use of mixing real learning steps in the world with simulated learning steps in an internal model. Experiments have shown this to be a very effective procedure when a forward model of the world (mapping from situations and actions into results) is available.

When the model is not made available *a priori*, it must also be constructed on line. Lin's results [Lin, 1991] show that using the model for learning while it is being constructed is not necessarily advantageous. This approach could probably be made more useful by adding a process that estimates the validity of the model, and only uses it for learning in situations in which it is valid.

4.2 Updating all Goals in DG Learning

A very simple kind of extra learning step can be directly added to DG learning. Whenever an action a is taken as a step from situation s to a goal g, we update the value of $DG(s, a, g)$ by looking ahead one step at the value of $DG(s', a', g)$ for the maximizing a'. This look-ahead process hinges on the relationship between s, a, and s'; the goal is relevant only for bookkeeping purposes.

We define *all goals* updating mode for DG learning to update $DG(s, a, g')$ for all $g' \in \mathcal{S}$, independent of what g was being sought when the action was taken. This updating mode requires an amount of work linear in the size of the set of situations; if this cost is prohibitive, a random or systematically-chosen subset of goals could be updated on each iteration.

The work that is done in updating all goals will not speed learning or performance for the particular goal that is being achieved; but it *will* result in an extremely efficient transfer of knowledge to the achievement of other goals. Singh [Singh, 1992] has also addressed this issue, but in a more complex network architecture.

4.3 Relaxation in G Learning

Because we know that the problem we are solving is one of finding shortest paths, we can apply basic knowledge about the nature of distance metrics: for any intermediate state s_i, the shortest path between states s_1 and s_2 is no longer than the the sum of the lengths of the shortest paths from s_1 to s_i and from s_i to s_2. Letting $D(s_1, s_2)$ be the length of the shortest path from s_1 to s_2, we can write this as

$$\forall s_i . D(s_1, s_2) \leq D(s_1, s_i) + D(s_i, s_2) \ \ .$$

The problem of finding shortest paths can be cast in terms of *relaxation*: starting with overestimates of D, except in cases where it is known, a relaxation step consists of an update of the form

$$D(s_1, s_2) := \min(D(s_1, s_2), D(s_1, s_i) + D(s_i, s_2)) \ \ .$$

If relaxations are performed at random, D will converge in the limit to the true distance measure. A much more useful method is provided by the Floyd-Warshall algorithm [Cormen *et al.*, 1990], which converges on the correct function after n^3 relaxations, where n is the size of the state set. It is written simply as follows:

```
for $i := 1$ to $n$
    for $j := 1$ to $n$
        for $k := 1$ to $n$
            $D(j, k) := \min(D(j, k), D(j, i) + D(i, k))$
```

It is crucial to iterate through the indices in the order specified, with the intermediate nodes varying most slowly.

Relaxations on the DG data structure are only slightly more complicated, due to the fact that actions are considered as well. A relaxation step consists of executing, for all $a \in \mathcal{A}$,

$$\begin{aligned} DG(s_1, a, s_2) \ := \ & \min(DG(s_1, a, s_2), \\ & DG(s_1, a, s_i) + \min_{a' \in \mathcal{A}} DG(s_i, a', s_2)). \end{aligned}$$

In the DG algorithm, relaxation steps may be freely interleaved with learning steps in the external world. Although relaxations may be chosen at random, it is much more efficient to run the Floyd-Warshall algorithm "in parallel" with the DG algorithm, performing the next few relaxation steps after every learning step. Because the DG values are being changed by the look-ahead process, it is useful to simply restart the Floyd-Warshall algorithm when it finishes. It might be desirable to detect convergence and quit doing relaxations after some point.

The ability to do relaxations of this type is very important. Basic temporal difference methods only allow value to be transferred back along the action paths one step at a time. But with these relaxations, if we know the distance from state s_1 to s_i and from s_i to s_2, we can immediately update our estimate of the distance from s_1 to s_2, with update time independent of the length of the path from s_1 to s_2.

5 Experimental Results

This section reports some preliminary experimental results comparing Q learning and DG learning in a popular synthetic domain.

5.1 Domain

The domain used in these experiments is similar to the one used by Sutton in his Dyna experiments. It can be thought of as a robot in a 10 by 10 grid of discrete locations. The robot has four actions, which nominally correspond to moving north, south, east, and west. In fact, when the robot executes one of these actions, it will only land on the nominal location with probability 0.2. The rest of the probability mass is divided evenly among the four locations directly neighboring the target location. The error distribution was chosen arbitrarily; another one could be easily substituted.

5.2 Q and DG Learning For a Single Goal

In the first set of experiments, the goal was placed at a single location and held there for the entire duration of the experiment. Whenever the agent reached the goal, the agent was "teleported" to a new location; learning data gathered on the teleportation step were discarded. Performance of agents was measured in terms of the average number of goals achieved per run of a given length.

The Q learning algorithm and the DG learning algorithm with single-goal updates were each run in this domain a number of times to determine appropriate parameter values. For both algorithms, T was chosen to be 0.1 and α was chosen to be 0.9. Each algorithm was then run for 20 runs of length 10,000 and the average number of goals was determined. For Q learning, the average was 324.85 and for DG learning it was 378.9. This difference is significant. An optimal strategy should be able to achieve goals in an average of about 11 steps.

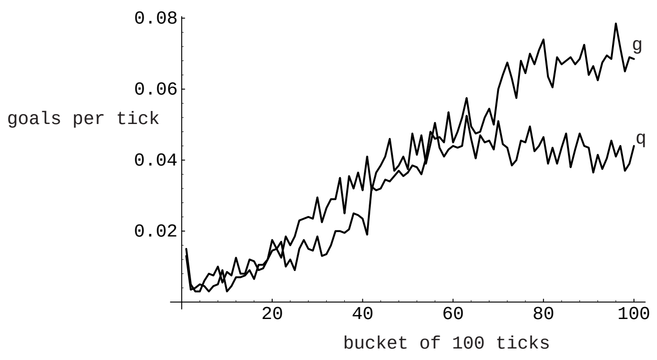

Figure 1: Learning curves of Q learning and DG learning for a single goal. Each curve is an average of 20 runs; each point represents the average number of goals achieved per tick, averaged over all the runs and over temporal stretches of 100 ticks.

Figure 1 shows the learning curves of the two algorithms. The Q learning algorithm is performing well until near the end of the run, where it seems to decrease in performance. It might be that a high learning rate or high temperature is responsible for this behavior; these parameter values were chosen on the basis of overall performance for runs of this length, but not necessarily for asymptotic performance. The performance of the DG algorithm is steadily increasing, and is approaching the optimal performance of 0.09 goals per step.

5.3 Q and DG Learning For Dynamic Goals

In the second experiment, we compared Q learning and DG learning on dynamic goals. In order to apply Q learning to this problem, we simply took the run-time goal specification to be an additional aspect of the situation description. The Q learning algorithm can be applied directly to this augmented state space.

The world was changed so that whenever the agent reached the goal, the goal was "teleported" at random to a new location. Again, the measure of success was the number of goals achieved over a fixed-length run. The parameters were held at the same values.

Each of the algorithms was run for 5 runs of length 20,000. The Q learning algorithm generated an average number of goals per step of 0.00536. The DG algorithm without extra updating was only slightly better with an average number of goals per step of 0.00619. When all-goals updating is added to the DG algorithm, we get a dramatic increase of performance, with an average number of 0.0427 goals per step. All of these differences are significant. The DG algorithm with all-goals updating learns how to achieve all goals simultaneously, making performance improve dramatically very early in the run. The learning curves are shown in figure 2.

5.4 Relaxation in DG Learning

Adding relaxation to DG learning with all-goals updating is difficult because of conflicting preferences of the basic DG learning method and the relaxation process. In general, methods like Q learning and DG learning

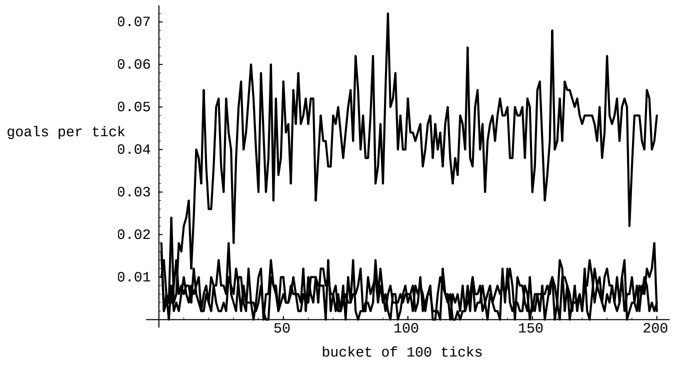

Figure 2: Learning curves of Q learning and DG learning for dynamic goals. Each curve is an average of 5 runs; each point represents the average number of goals achieved per tick, averaged over all the runs and over temporal stretches of 100 ticks. The top curve shows the performance of DG learning with all-goals updating. The bottom two curves are largely indistinguishable; they show the performance of DG learning with regular updating and of Q learning.

work best when the initial values are optimistic (that is, high for Q learning and low for DG learning); this leads to extra exploration early in the run. On the other hand, relaxation requires that the DG values be pessimistic (too high), so that taking the minimum does not propagate errors.

It has proven difficult in practice to satisfy both of these constraints. In further studies, we will attempt to solve this problem through the use of the interval estimation method. In this method, a confidence interval estimate is constructed for the underlying DG values. The action whose confidence interval has the highest upper bound is chosen for execution. This naturally causes exploration for actions that have not been chosen very often, but over time the exploration dies away. Having confidence intervals gives us a natural way of providing overestimates of the distances for use in the relaxation steps, as well.

6 Increasing Efficiency through Hierarchy

The DG learning algorithm shows increased performance as a function of the number of learning steps the agent performs in the world. This performance comes at a penalty in storage space and in computational time per learning step. The DG values require $O(|\mathcal{S}|^2|\mathcal{A}|)$ space, and all-goals updating requires $O(|\mathcal{S}||\mathcal{A}|)$ time per tick.

All of this work allows us to learn optimal paths between arbitrary points in our state space. It is rarely the case, however, that optimal behavior is actually necessary, and it might be advantageous to trade some degree of performance for increased computational efficiency.

In work that builds upon the ideas in this paper, Kaelbling [Kaelbling, 1993a] introduces a hierarchical model of goal achievement that allows efficient learning of approximately optimal paths. The hierarchical model can be informally described by analogy with our freeway system: we plan a route from our current location to the nearest freeway on-ramp, plan a series of legs on freeway segments, then plan a route from the freeway off-ramp to our destination. Such a hierarchical structure requires learning of paths only between local states at the lowest level, and requires learning of the higher level structure. A preliminary implementation shows that this notion has considerable promise.

7 Conclusions

Many problems considered in AI and robotics have the nature of dynamically changing goals of achievement. A re-application of the ideas of on-line stochastic dynamic programming [Barto *et al.*, 1991] to the problem of calculating shortest paths results in DG learning, which is a very effective learning method. It is especially valuable for its ability to transfer knowledge between tasks

The studies reported on here used tables to represent the DG and Q values; there is even more opportunity to study between-task transfer by using function-approximation methods that generalize over similar instances. An interesting area for future work lies in the integration of DG learning with various artificial neural network models of association.

References

[Barto *et al.*, 1989] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and sequential decision making. Technical Report 89-95, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1989. Also published in *Learning and Computational Neuroscience: Foundations of Adaptive Networks*, Michael Gabriel and John Moore, editors. The MIT Press, Cambridge, Massachusetts, 1991.

[Barto *et al.*, 1991] Andrew G. Barto, Steven J. Bradtke, and Satinder Pal Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report 91-57, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1991.

[Bellman, 1957] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[Cormen *et al.*, 1990] Thomas H. Cormen, Charles E. Leiserson, and Ronals L. Rivest. *Introduction to Algorithms.* The MIT Press / McGraw Hill, Cambridge, Massachusetts, 1990.

[Howard, 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes.* The MIT Press, Cambridge, Massachusetts, 1960.

[Kaelbling, 1993a] Leslie Pack Kaelbling. Hierarchical learning: Preliminary results. In *Proceedings of the Tenth International Conference on Machine Learning*, Amherst, Massachusetts, 1993. Morgan Kaufmann.

[Kaelbling, 1993b] Leslie Pack Kaelbling. *Learning in Embedded Systems.* The MIT Press, Cambridge, Massachusetts, 1993. Also available as a PhD Thesis from Stanford University, 1990.

[Lin, 1991] Long-Ji Lin. Self-improving based on reinforcement learning, planning, and teaching. In *Proceedings of the Eighth International Workshop on Machine Learning*, Evanston, Illinois, 1991. Morgan Kaufmann.

[Singh, 1992] Satinder Pal Singh. Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3):323–340, 1992.

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

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

[Thrun, 1992] Sebastian B. Thrun. The role of exploration in learning control. In David A. White and Donald A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches.* Van Nostrand Reinhold, New York, 1992.

[Watkins, 1989] C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, 1989.

[Whitehead and Ballard, 1989] 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*, pages 354–357, Ithaca, New York, 1989. Morgan Kaufmann.