As appeared in the Proceedings of the International Workshop on the Simulation of Adaptive Behavior: From Animals to Animats, MIT Press, 1991, pp. 288-296.

Reinforcement Learning Architectures for Animats

Richard S. Sutton
GTE Laboratories Incorporated
Waltham, MA 02254
sutton@gte.com

Abstract

In the first part of this paper I argue that the learning problem facing animats is essentially that which has been studied as the reinforcement learning problem—the learning of behavior by trial and error without an explicit teacher. A brief overview is presented of the development of reinforcement learning architectures over the past decade, with references to the literature.

The second part of this paper presents Dyna, a class of architectures based on reinforcement learning but which go beyond trial-and-error learning. Dyna architectures include a learned internal model of the world. By intermixing conventional trial and error with hypothetical trial and error using the world model, Dyna systems can plan and learn optimal behavior very rapidly. Results are shown for simple Dyna systems that learn from trial and error while they simultaneously learn a world model and use it to plan optimal action sequences. We also show that Dyna architectures are easy to adapt for use in changing environments.

1 Animats and the Reinforcement Learning Problem

What is an Animat? An animat is an adaptive system designed to operate in a tight, closed-loop interaction with its environment. An animat need not be a learning system, but often it is; some sense of adaptation of behavior to variations in the environment is required.

Figure 1 is a representation of the animat problem as I see it. On some short time cycle, the animat receives sensory information from the environment and chooses an action to send to the environment. In addition, the animat receives a special signal from the environment called the *reward*. Unlike the sensory information, which may be a large feature vector, or the action, which may also have many components, the reward is a single real-valued scalar, a number. The goal

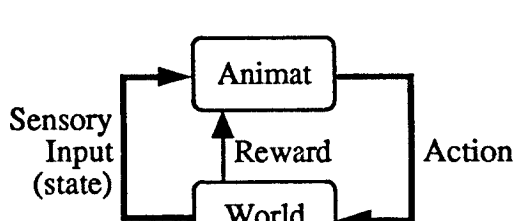


Figure 1. The Reinforcement Learning Problem facing an Animat. The goal is to maximize cumulative reward.

of adaptation is the maximization of the cumulative reward received over time.

This formulation of the animat problem is the same as that widely used in the study of *reinforcement learning*. In fact, reinforcement learning systems can be defined as learning systems designed for and that perform well on the animat problem as described above. Informally, we define reinforcement learning as learning by trial and error from performance feedback—i.e., from feedback that evaluates the behavior generated by the animat but does not indicate correct behavior. In the next section we briefly survey reinforcement learning architectures.

One might object to the problem formulation in Figure 1 on the grounds that all possible goals have been reduced to a scalar reward. Although this appears limiting, in practice it has proved to be a useful way of structuring the problem. Some examples of goals formulated in this way are:

- Foraging: Reward is positive for finding food objects, negative for energetic motion, slightly negative for standing still.
- Pole-balancing (balancing a pole by applying forces to its base): The reward is zero while the pole is balanced, and then becomes -1 if the pole falls over or if the base moves too far out of bounds.
- Towers of Hanoi: Reward is positive for reaching the goal state.

- Recycling Robot: Reward is positive for dropping soda cans in the recycling bin, negative for bumping into things, more negative for bumping hard into things, most negative for being yelled at or for running down the battery, etc.

- Video Game Playing: One unit of reward for every point scored.

Another reason one might object to the problem formulation in Figure 1 is that the goal of learning is defined solely in terms of something (the reward) arising from the external environment, not from the animat itself. Often goals do concern the evolution of the the animat's own internal state, e.g., its energy reservoirs. Fortunately, it appears that most goals, perhaps all, can be put in the "external reward" form simply by redrawing the boundary between animat and environment. For example, if the goal concerns the animat's energy reservoirs, then these are considered part of the environment; if the goal concerns the positions of the animat's limbs, then these too can be considered part of the environment—the animat's boundary is drawn at the interface between the limbs and their control systems. Roughly speaking, things are considered part of the animat if they are completely, directly, and with certainty controllable; things are considered part of the environment if they are not. Since the goal is always something over which we have imperfect or uncertain control, it is placed outside the animat.

2 Overview of Reinforcement Learning Architectures

In this section we review the major steps in the development of reinforcement learning architectures over the last decade. These steps are illustrated by the four architectures shown in Figure 2.

All reinforcement learning involves the learning of a mapping from a representation of a situation or *state* to an appropriate *action* (or a probability distribution over actions) for that situation. This mapping is called the *policy*; it specifies what the animat will do in each situation at its current stage of learning. The simplest reinforcement learner that one might imagine, then, would consist only of a policy and a way of adjusting it based on reward, as shown in Figure 2a. Such architectures, in which the policy is the only modifiable data structure (and, indeed, the only structure at all) are here called *policy-only* architectures.

The policy can be implemented in any of a number of ways. It can be a connectionist neural network, or a symbolic learning structure such as a decision tree or lisp program, or a conventional set of statistics such as is used by maximum-likelihood or nearest-neighbor techniques. Any of these methods can be used—with varying advantages and disadvantages—to implement this and the other modifiable structures (in the other architectures) shown in Figure 2.

The learning algorithm for the policy must be of a slightly unusual type. Standard supervised learning methods such as backpropagation are not sufficient here, but must be modified, at least slightly, to take into account the fact that a "target" action is not directly available. Instead, a form of correlation must be done between the reward received and the actions taken by the animat, all with respect to the sensory input. Actions correlated with high reward have their probability of being repeated increased, while those correlated with low reward have their probability decreased. Examples of such algorithms for policy-only architectures can be found in (Farley & Clark, 1954; Widrow, Gupta & Maitra, 1973; Barto & Sutton, 1981; Barto & Anandan, 1985).

Policy-only architectures really only work well when it is clear a priori what constitutes a high reward and what constitutes a low one—for example, if all high rewards are positive and all low rewards are negative. Often, however, rewards are not distributed around a baseline of zero, but around some other, unknown value. Worse yet, the baseline may change from state to state. A low reward value in one state may be the highest attainable in another. To handle such variations, a baseline value must be learned that is a function of the state; the actual reward is then compared with the current state's baseline. This is what is done in *reinforcement-comparison* architectures (Figure 2b). As a baseline, these architectures usually use a *prediction* of the reward. The prediction error—the difference between predicted and actual reward—is used both as an enhanced, zero-balanced reward signal for adjusting the policy *and* as an ordinary error for learning the reward predictions (Figure 2b). A variety of reinforcement-comparison algorithms have been explored and compared (Barto, Sutton & Brouwer, 1981; Sutton, 1984; Williams & Peng, 1989; Williams, 1986).

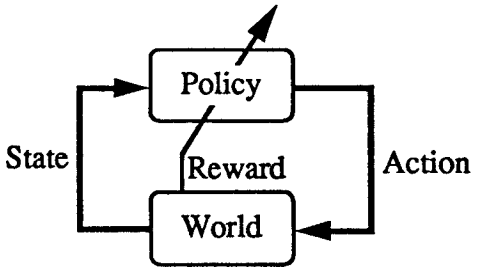
Reinforcement comparison architectures are effective at optimizing immediate rewards, but not at optimizing total reward in the long run. The problem is that actions have two kinds of consequences—they affect the next reward *and* they affect the next state, but reinforcement-comparison architectures only take the first of these into account. Suppose an action produces high immediate reward but deposits the environment in a state from which only low reward can be obtained? In order to optimize long-term reward, these delayed affects of action must be taken into account.

The *adaptive heuristic critic (AHC)* architecture, shown in Figure 2c, was designed to take such delayed effects into account. The predictor of immediate reward has been replaced with a predictor of *return*, a measure of long-term cumulative reward. For any state x, the return is formally defined as the expected value of the sum of all future rewards, discounted by their delay, given that the systen starts in x:

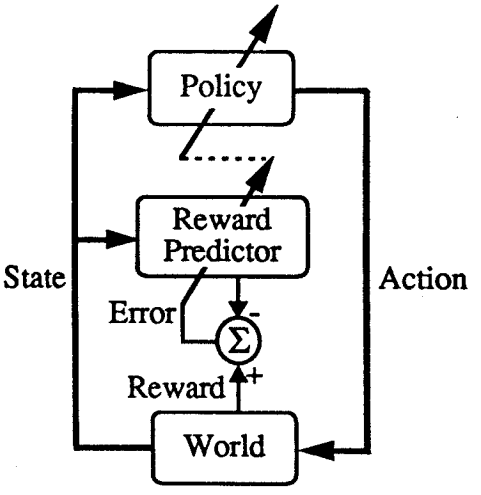
$$return(x) = E\left\{ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| x_0 = x \right\},$$

where γ, $0 \leq \gamma \leq 1$, is the discount rate determining how fast one's concern for delayed reward falls off with length of the delay.[1] The "return predictor" box

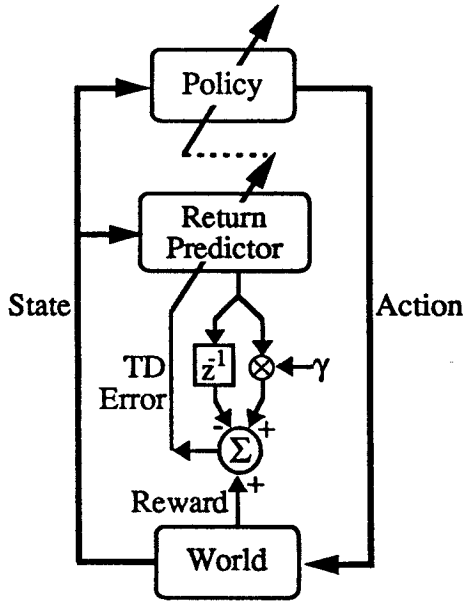
[1] This is analogous to the discount rate in economics—a



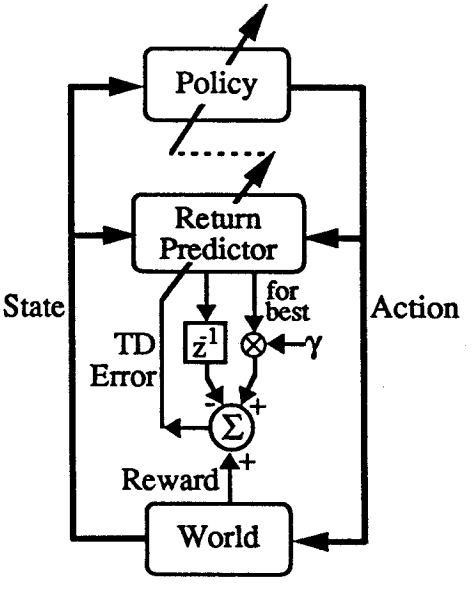
A) Policy-Only



B) Reinforcement Comparison



C) Adaptive Heuristic Critic



D) Q-Learning

Figure 2. Overview of Reinforcement Learning Architectures. See text. The symbol $\boxed{z^{-1}}$ indicates a one time step delay, and the symbol $\otimes$ indicates multiplication. The line labeled "for best" in D is the prediction of return for the best action; the other output from the return predictor is the prediction of return for the action actually taken.

in Figure 2c comes to predict this return by virtue of the circuit shown below it for calculating a temporal-difference error (Sutton, 1988). In all other respects, the learning algorithm inside this box could be exactly the same as that used in the "reward predictor" box of Figure 2b. The AHC architecture has been used in a variety of learning control tasks (Sutton, 1984; Barto, Sutton & Anderson, 1983; Anderson, 1987; Barto, Sutton & Watkins, 1989).

Finally, Figure 2d shows the most recent reinforcement learning architecture, *Q-learning* (Watkins, 1989). The primary innovation here is that the predicted return is now a function of action as well as state. Formally, the return for a state x and action a is defined as

$$return(x,a) = E\left\{\sum_{t=0}^{\infty}\gamma^t r_{t+1}\middle| x_0 = x, a_0 = a\right\}. \quad (1)$$

Two kinds of return are always predicted for the current state. One is the best predicted return for the state—the predicted return for the action with the highest predicted return. The other is the predicted return for the action actually selected. These two predictions are then combined according to the same circuit as used in the AHC architecture. In Q-learning, the policy could be a separate modifiable data structure as suggested by Figure 2d, but most often it is simply a dependent function of the return predictions. For example, the policy may be simply to pick the action that obtains the maximal return prediction. A particular Q-learning algorithm will be given in detail later.

3 Dyna Architectures

Reinforcement learning architectures are effective at trial-and-error learning, but no more. They can not do any of the things that are considered "cognitive," such as reasoning or planning. They do not learn an internal model of the world's dynamics, of what-causes-what, but only of what-to-do (policy) and how-well-am-I-doing (return predictions). This is an important limitation because potentially much more can be learned in the form of a world model than can be learned by trial and error; the reward signal is just a scalar, while the sensory input signal is a much richer potential source of training information. And what if the goal changes? Typically, a world model can remain relatively intact over goal changes and can assist in achieving the new goal, whereas policy and return predictions must be totally changed.

Dyna architectures are simple extensions of reinforcement learning architectures to include an internal world model (Sutton, 1990; Whitehead, 1989). The world model is defined as something that behaves like the world: given a state and an action it is supposed to output a prediction of the resultant reward and next state. If the world's state is observable, then it is straightforward to learn a world model using supervised learning methods and training examples taken from actual interactions with the world. If the world is not observable, then it must be inferred estimated from the history of sensory input and action. Although there are a variety of algorithms for doing this, in the general case it is still an open problem. Here we assume for the time being that the state is observable.

1. Observe the current state x and choose an action: $a \leftarrow Policy(x)$.
2. Send the action to the world and observe the resultant next state y and reward r.
3. Apply a reinforcement learning method to the experience x, y, a, and r.
4. Update the world model based on the experience x, y, a, and r.
5. Repeat the following steps k times:
   - 5.1 Select a hypothetical state x and hypothetical action a.
   - 5.2 Send x and a to the world model and obtain predictions of next state y and reward r.
   - 5.3 Apply a reinforcement learning method to the hypothetical experience x, y, a, and r.
6. Go to 1.

Figure 3. Main Loop of the Dyna Algorithm.

In Dyna architectures the model is used as a direct replacement for the world in one of the reinforcement learning architectures shown in Figure 2. Reinforcement learning continues in the usual way, but, in addition, learning steps are also run using the model in place of the world, using predicted outcomes rather than actual ones. For each real experience with the world, many hypothetical experiences generated by the world model can also be processed and learned from. The cumulative effect of these hypothetical experiences is that the policy approaches the optimal policy given the current model; a form of planning has been achieved. The overall Dyna algorithm is given in Figure 3.

Figure 4 shows results for a Dyna architecture based on the AHC architecture, called *Dyna-AHC*. The task is to navigate through the maze shown in Figure 4 from the starting state "S" to the goal state "G". From each state there are four possible actions: UP, DOWN, RIGHT, and LEFT, which change the state accordingly, except where such a movement would take the system into a barrier (shaded state) or outside the maze, in which case the state is not changed. Reward is zero for all transitions except for those into the goal state, for which it is +1. The lower left portion of the figure shows learning curves for Dyna-AHC systems with $k = 100$, $k = 10$, and $k = 0$. The $k = 0$ case involves no hypothetical steps; this is a pure trial-and-error reinforcement-learning system. Although the

dollar today is worth more than a dollar tomorrow.

length of path taken from start to goal falls dramatically for this case, it falls much *more* rapidly for the cases including hypothetical experiences, showing the benefit of planning using the learned world model. For $k = 100$, the optimal path was generally found and followed by the fourth trip from start to goal. This is extremely rapid learning.

Figure 5 shows why the Dyna-AHC system solved this problem so much faster than the pure reinforcement learning system. Shown are the policies found by the $k = 0$ and $k = 100$ systems half-way through the second trial. Without planning ($k = 0$), each trial adds only one additional step to the policy, and so only one step (the last) has been learned so far. With planning, the first trial also learned only one step, but here during the second trial an extensive policy has been developed that by the trial's end will reach back almost to the start state. By the end of the third or fourth trial a complete optimal policy will have been found and perfect performance attained.

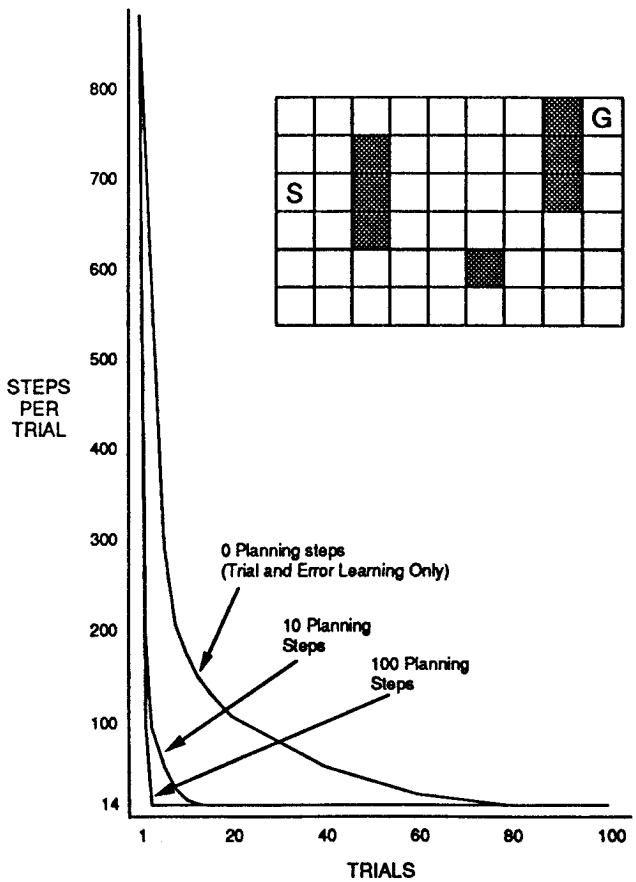


Figure 4. Learning Curves for Dyna-AHC Systems on a Simple Navigation Task. A trial is one trip from the start state "S" to the goal state "G". The more hypothetical experiences ("planning steps") using the world model, the faster an optimal path was found. These data are averages over 100 runs.

4 Problems of Changing Worlds

Suppose that, after a Dyna-AHC system has learned the optimal path from start to goal, a new barrier is added that blocks the optimal path. The Dyna-AHC system discussed above will run into the block and then try the newly ineffective action many hundreds of times. Eventually, the correct new path will be found, but the process is very slow. It seems inappropriately slow in that the system's world model is updated immediately. Even though the world model knows that the formerly good action is now poor, this is not reflected in the system's behavior for a long time. I call this the *blocking problem.*

Now consider a second sort of change in the environment. Suppose, after the optimal path has been learned, a barrier is removed that permits a shorter path from start to goal. The Dyna-AHC system is unable to take advantage of such a shortcut; it never wavers from the formerly optimal path and thus never discovers that the former obstacle is gone. I call this the *shortcut problem.* In seeking to improve the Dyna-AHC system to handle blocks, one might also seek to improve it to handle shortcuts. What is needed here is some way of continually testing the world model. In the next section we present a Dyna architecture based on Q-learning that handles both kinds of changes with little increase in complexity.

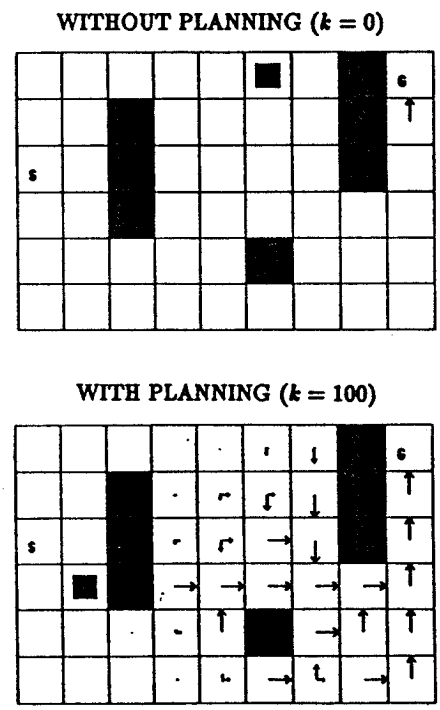


Figure 5. Policies Found by Planning and Non-Planning Dyna-AHC Systems by the Middle of the Second Trial. The black square indicates the current location of the Dyna system. The arrows indicate action probabilities (excess over the smallest) for each direction of movement.

5 Dyna-Q Architectures

In the rest of this paper we focus on Dyna architectures based on Q-learning. In this section we give a complete specification of two such Dyna-Q architectures, called *Dyna-Q-* and *Dyna-Q+*.

Recall that, in Q-learning, the return predictions are made on the basis of both a state and an action (Figure 2d). For any state x and action a, let Q_{xa} be the current prediction of $return(x,a)$; these are also called *Q-values*. Suppose the experience x, y, a, and r (state, next state, action, and reward) occurs. The simplest form of Q-learning is based on constructing from this the following training example:

$$Q_{xa} \quad \text{should be} \quad r + \gamma \max_{b \in A} Q_{yb}, \qquad (2)$$

where A is the set of actions and γ is the discount-rate parameter from (1). This training example could be passed to any appropriate supervised learning algorithm. In the results reported in the next section, the return predictor was implemented as a table. The complete update rule for this case is:

$$Q_{xa} \leftarrow Q_{xa} + \beta(r + \gamma \max_{b \in A} Q_{yb} - Q_{xa}),$$

where β is a positive learning-rate parameter. This is the only update rule in the version of Q-learning used here. The policy was to deterministically select the action with highest Q-value:

$$Policy(x) = \arg\max_{a \in A} Q_{xa}.$$

The Dyna-Q algorithms use Q-learning as their reinforcement learning method in Steps 3 and 5.3 of Figure 3. In the experiments presented in the next section, the starting states and actions for hypothetical experiences were selected uniformly randomly (Step 5.1). The model was implemented as a lookup table with an entry for each state-action pair. This completes the description of Dyna-Q-.

As we show below, Dyna-Q- is not able to solve the shortcut problem. To help on this problem, Dyna-Q+ uses an additional memory structure to keep track of the degree of uncertainty about each component of the model. For each state x and action a, a record is kept of the number of time steps n_{xa} that have elapsed since a was tried in x in a real experience. The square root $\sqrt{n_{xa}}$ is used as a measure of the uncertainty about Q_{xa}.[2] To encourage exploration, each state-action pair is then given an *exploration bonus* proportional to this uncertainty measure. For real experiences, the policy is to select the action a that maximizes $Q_{xa} + \epsilon\sqrt{n_{xa}}$, where ϵ is a small positive parameter. This method of encouraging variety is very similar to that used in Kaelbling's (1990) interval-estimation algorithm.

[2]The use of the square root is heuristic but not arbitrary, as the standard deviation of stationary, cumulative random processes increases with the square root of the number of cumulating steps.

However, this approach alone does not take advantage of the planning capability of Dyna architectures. Suppose there is a state-action pair that has not been tested in a long time, but that is far from the currently preferred path, and thus extremely unlikely to be tried even with the exploration bonus discussed above. In a Dyna system, why not expect the system to *plan* an action sequence to go out and test the uncertain state-action pair? If there is genuine uncertainty, then there is potential benefit in going out and trying the action, and thus forming such a plan is simply rational behavior and should be done. It turns out that there is a simple way to do this in Dyna. The exploration bonus of $\epsilon\sqrt{n_{xa}}$ is used not only in the policy, but also in the update equation for the Q-values. That is (2) is replaced by:[3]

$$Q_{xa} \quad \text{should be} \quad r + \epsilon\sqrt{n_{xa}} + \gamma \max_{b \in A} Q_{yb}.$$

In addition, the system is permitted to hypothetically experience actions it has never before tried, so that the exploration bonus for not having tried them can be propagated back by the planning process. This can be done by starting the system with a non-empty initial model. In the experiments with Dyna systems reported below, actions that had never been tried were assumed to produce zero reward and leave the state unchanged. This completes the specification of Dyna-Q+.

6 Changing-World Experiments

Experiments were performed to test the ability of Dyna systems to solve blocking and shortcut problems. All three Dyna systems were used: Dyna-AHC, Dyna-Q-, and Dyna-Q+. All systems used $k = 10$. For the Dyna-AHC system, the other parameters were set as in the navigation experiment. For the Dyna-Q systems, they were set at $\beta = 0.5$, $\gamma = 0.9$, and $\epsilon = 0.001$. In all cases, the initial Q-values were zero.

The blocking experiment used the two mazes shown in the upper portion of Figure 6. Initially a short path from start to goal was available (first maze). After 1000 time steps, by which time the short path was usually well learned, that path was blocked and a longer path was opened (second maze). Performance under the new condition was measured for 2000 time steps. Average results over 50 runs are shown in Figure 6 for the three Dyna systems. The graph shows a *cumulative* record of the number of rewards received by the system up to each moment in time. In the first 1000 trials, all three Dyna systems found a short route to the goal, although the Dyna-Q+ system did so significantly faster than the other two. After the short path was blocked at 1000 steps, the graph for the Dyna-AHC system remains almost flat, indicating that it was unable to obtain further rewards. The Dyna-Q systems, on the other hand, clearly solved the blocking problem, reliably finding the alternate path after about 800 time steps.

[3]Note that this differs from (2) only on hypothetical experiences, as $n_{xa} = 0$ on real experiences.

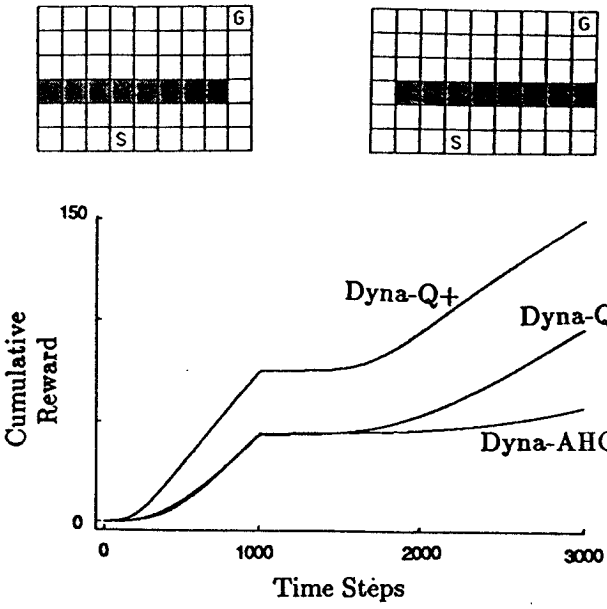


Figure 6. Average Performance of Dyna Systems on a Blocking Task. The left maze was used for the first 1000 time steps, the right maze for the last 2000. Shown is the cumulative reward received by a Dyna system at each time (e.g., a flat period is a period during which no reward was received).

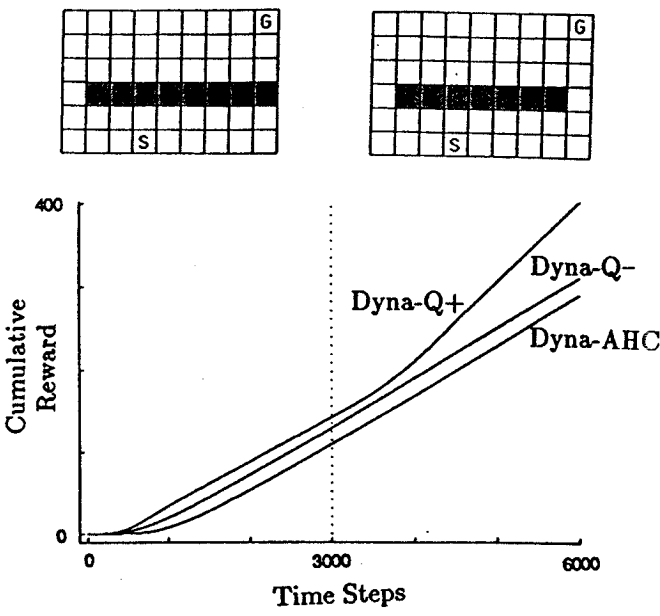


Figure 7. Average Performance of Dyna Systems on a Shortcut Task. The left maze was used for the first 3000 time steps, the right maze for the last 3000. Shown is the cumulative reward received by a Dyna system at each time (e.g., the slope corresponds to the *rate* at which reward was received).

The shortcut experiment began with only a long path available (first maze of Figure 7). After 3000 times steps all three Dyna systems had learned the long path, and then a shortcut was opened without interfering with the long path (second maze of Figure 7). The lower part of Figure 7 shows the results. The increase in the slope of the curve for the Dyna-Q+ system, while the others remain constant, indicates that it alone was able to find the shortcut. The Dyna-Q+ system also learned the original long route faster than the Dyna-Q- system, which in turn learned it faster than the Dyna-AHC system. However, the ability of the Dyna-Q+ system to find shortcuts does not come totally for free. Continually re-exploring the world means occasionally making suboptimal actions. If one looks closely at Figure 7, one can see that the Dyna-Q+ system actually acheives a slightly lower *rate* of reinforcement during the first 3000 steps. In a static environment, Dyna-Q+ will eventually perform worse than Dyna-Q-, whereas, in a changing environment, it will be far superior, as here. One possibility is to use a meta-level learning process to adjust the exploration parameter ϵ to match the degree of variability of the environment.

One strength of the Dyna approach is that it applies to stochastic problems as well as deterministic ones. We have explored this direction in recent work, but are not yet ready to present systematic results. The basic idea is to learn a model which predicts not a determinstc next state and next reward, but rather a probability distribution over next states and next rewards. In the simple cases we have explored, this reduces to counting the number of times each possible outcome has occurred. In hypothetical experiences, the expected value of the right of (2) is then estimated using the sample statistics. A slightly different exploration bonus is also needed. Promising preliminary results have so far been obtained for simple problems involving random autonomous agents and stochastic state transitions (e.g., action UP takes the system to the state above 80% of the time, and to a random neighboring state 20% of the time).

Further results are needed for a thorough comparison of Dyna-AHC and Dyna-Q architectures, but the results presented here suggest that it is easier to adapt Dyna-Q architectures to changing environments.

7 Limitations and Conclusions

The simple illustrations presented here are clearly limited in many ways. The state and action spaces are small and denumerable, permitting tables to be used for all learning processes and making it feasible for the entire state space to be explicitly explored. For large state spaces it is not practical to use tables or to visit all states; instead one must represent a limited amount of experience compactly and generalize from it. Both Dyna architectures are fully compatible with the use of a wide range of learning methods for doing this. For example, Lin (1990) has explored the use of Dyna architectures using backpropagation networks instead of tables.

We have also assumed that the Dyna systems have

explicit knowledge of the world's state. In general, states can not be known directly, but must be estimated from the pattern of past interaction with the world (Rivest & Schapire, 1987; Mozer and Bachrach, 1990). Dyna architectures can use state estimates constructed in any way, but will of course be limited by their quality and resolution. A promising area for future work is the combination of Dyna architectures with egocentric or "indexical-functional" state representations (Agre & Chapman, 1987; Whitehead, 1989).

Yet another limitation of the Dyna systems presented here is the trivial form of search control used. Search control in Dyna boils down to the decision of whether to consider hypothetical or real experiences, and of picking the order in which to consider hypothetical experiences. The tasks considered here are so small that search control is unimportant, and thus it was done trivially, but a wide variety of more sophisticated methods could be used. Particularly interesting is the possibility of using Dyna architectures at higher levels to make these decisions.

Finally, the examples presented here are limited in that reward is only non-zero upon termination of a path from start to goal. This makes the problem more like the kind of search problem typically studied in AI, but does not show the full generality of the framework, in which rewards may be received on any step and there need not even exist start or termination states.

Despite these limitations, the results presented here are significant. They show that the use of an internal model can dramatically speed trial-and-error learning processes even on simple problems. Moreover, they show how planning can be done with the incomplete, changing, and oftimes incorrect world models that are contructed through learning. Finally, they show how the functionality of planning can be obtained in a completely incremental manner, and how a planning process can be freely intermixed with execution and learning. I conclude that it is not necessary to choose between planning, reacting, and learning. These three can be integrated not only into one animat, but into a single algorithm, where each appears as a different facet of that algorithm.

Acknowledgments

The author gratefully acknowledges the extensive contributions to the ideas presented here by Andrew Barto, Chris Watkins and Steve Whitehead. I also wish to also thank the following people for ideas and discussions: Paul Werbos, Luis Almeida, Ron Williams, Glenn Iba, Leslie Kaelbling, John Vittal, Charles Anderson, Bernard Silver, Oliver Selfridge, Judy Franklin, Tom Dean and Chris Matheus.

References

Agre, P. E., & Chapman, D. (1987) Pengi: An implementation of a theory of activity. *Proceedings of AAAI-87*, 268–272.

Anderson, C. W. (1987) Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning*, 103–114. Morgan Kaufmann, Irvine, CA.

Barto, A. G., & Anandan, P. (1985) Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics 15*, 360–375.

Barto, A. G., & Sutton. R. S. (1981) Landmark learning: An illustration of associative search. *Biological Cybernetics 42*, 1–8.

Barto, A. G., Sutton R. S., & Anderson, C. W. (1983) Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics 13*, 834–846.

Barto, A. G., Sutton, R. S., & Brouwer, P. S. (1981) Associative search network: A reinforcement learning associative memory. *Biological Cybernetics 40*, 201–211.

Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989) Learning and sequential decision making. COINS Technical Report 89-95, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA 01003. Also to appear in *Learning and Computational Neuroscience*, M. Gabriel and J.W. Moore (Eds.), MIT Press, 1991.

Farley, B. G., & Clark, W. A. (1954) Simulation of self-organizing systems by digital computer. *I.R.E. Transactions on Inf. Theory 4*, 76–84.

Kaelbling, L. (1990) *Learning in Embedded Systems*. Stanford Computer Science Ph.D. Dissertation. Technical Report No. TR-90-04, Teleos Research, Palo Alto, CA.

Lin, L. (1990) Self-improving reactive agents: Case studies of reinforcement learning frameworks. *SAB-90*.

Mozer, M. C., & Bachrach, J. (1990) Discovering the structure of a reactive environment by exploration. In *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan Kaufmann, San Mateo, CA. See also Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado at Boulder 80309.

Rivest, R. L., & Schapire, R. E. (1987) A new approach to unsupervised learning in deterministic environments. *Proceedings of the Fourth International Workshop on Machine Learning*, 364–375. Morgan Kaufmann, Irvine, CA.

Sutton, R. S. (1984) Temporal credit assignment in reinforcement learning. Doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

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

Sutton, R. S. (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *Proceedings of the Seventh International Conference on Machine Learning*, 216–224, Morgan-Kaufmann.

Watkins, C. J. C. H. (1989) *Learning with Delayed Rewards.* Ph.D. dissertation, Cambridge University, Psychology Department.

Widrow, B., Gupta, N. K., & Maitra, S. (1973) Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE Transactions on Systems, Man, and Cybernetics 5*, 455–465.

Whitehead, S. D. (1989) Scaling reinforcement learning systems. Technical Report 304, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.

Williams, R. J., & Peng, J. (1989) Reinforcement learning algorithms as function optimizers. *Int. Joint Conference on Neural Networks.*

Williams, R. J. (1986) Reinforcement learning in connectionist networks: A mathematical analysis, University of California, San Diego Inst. for Cognitive Science Technical Report 8605.