# 8 First Results with Dyna, an Integrated Architecture for Learning, Planning and Reacting

Richard S. Sutton

## 8.1 Introduction

How should a robot decide what to do? The traditional answer in AI has been that it should deduce its best action in light of its current goals and world model, i.e., that it should *plan*. However, it is now widely recognized that planning's usefulness is limited by its computational complexity and by its dependence on a complete and accurate world model. An alternative approach is to do the planning in advance and compile its result into a set of rapid *reactions*, or situation-action rules, which are then used for real-time decision making. Yet a third approach is to *learn* a good set of reactions by trial and error; this has the advantage of eliminating the dependence on a world model. In this chapter I briefly introduce *Dyna*, a simple architecture integrating and permitting tradeoffs among these three approaches. Results are presented for a simple Dyna system that learns from trial and error while it learns a world model and uses the model to plan reactions that result in optimal action sequences.

Dyna is based on the old idea that planning is like trial-and-error learning from hypothetical experience (Craik 1943, Dennett 1978). The theory of Dyna is based on the classical optimization technique of *dynamic programming* (Bellman 1957, Ross 1983) and on the relationship of dynamic programming to reinforcement learning (Watkins 1989, Barto, Sutton, and Watkins 1989), to temporal-difference learning (Sutton 1988), and to AI methods for planning and search. Werbos (1987) has previously argued for the general idea of building AI systems that approximate dynamic programming, and Whitehead (1989) and others (Sutton and Barto 1981, Sutton and Pinette 1985, see also Rumelhart et al. 1986) have presented results for the specific idea of augmenting a reinforcement learning system with a world model used for planning.

The Dyna architecture consists of four primary components, interacting as shown in figure 8.1. The *policy* is simply the function formed by the current set of reactions; it receives as input a description of the current state of the world and produces as output an action to be sent to the world. The *world* represents the task to be solved; prototypically it is the robot's external environment. The world receives actions from the policy and produces a next state output and a reward output. The overall task is defined as maximizing the long-term average reward per time step (cf. Russell 1989). The Dyna architecture also includes an explicit *world model.* The world model is intended to mimic the one-step input-output behavior of the real world. Finally, the Dyna architecture includes an *evaluation function* that rapidly maps states to values, much as the policy rapidly maps states to actions. The evaluation function, the policy, and the world model are each updated by separate learning processes.

For a fixed policy, Dyna is a simple reactive system. However, the policy is continually adjusted by an integrated planning/learning process. The policy is, in a sense, a *plan*, but one that is completely conditioned by current input. The planning process is incremental and can be interrupted and resumed at any time. It consists of a series of shallow searches, each typically of one ply, and yet ultimately produces the same result as an arbitrarily deep conventional search. I call this *relaxation planning.* Dynamic programming is a special case of this.

Relaxation planning is based on continuously adjusting the evaluation function in such a way that credit is propagated to the appropriate steps within action sequences. Generally speaking, the evaluation of a state $x$ should be equal to the best of the states $y$ that can be reached from it in one action, taking into consideration the reward (or cost) $r$ for that one transition, i.e.:

$$Eval(x) \text{ “=” } \max_{a \in Actions} E\{r + Eval(y) \mid x, a\}, \tag{8.1}$$

where $E\{\cdot \mid \cdot\}$ denotes a conditional expected value and the equal sign is quoted to indicate that this is a condition that we would like to hold, not one that necessarily does hold. If we have a complete model of the world, then the right-hand side can be computed by looking ahead one action. Thus, we can generate any number of training examples for the process that learns the evaluation function: for any $x$, the right-hand side of equation 8.1 is the desired output. If the learning process converges such that equation 8.1 holds in all states, then the optimal policy is given by choosing the action in each state $x$ that achieves the maximum on the right-hand side. There is an extensive theoretical basis from dynamic programming for algorithms of this type for the special case in which the evaluation function is tabular, with enumerable states and actions. For example, this theory guarantees convergence to a unique evaluation function satisfying equation 8.1 and that the corresponding policy is optimal (e.g., see Ross 1983).

The evaluation function and policy need not be tables, but can be more compact function approximators such as decision trees, *k-d* trees,

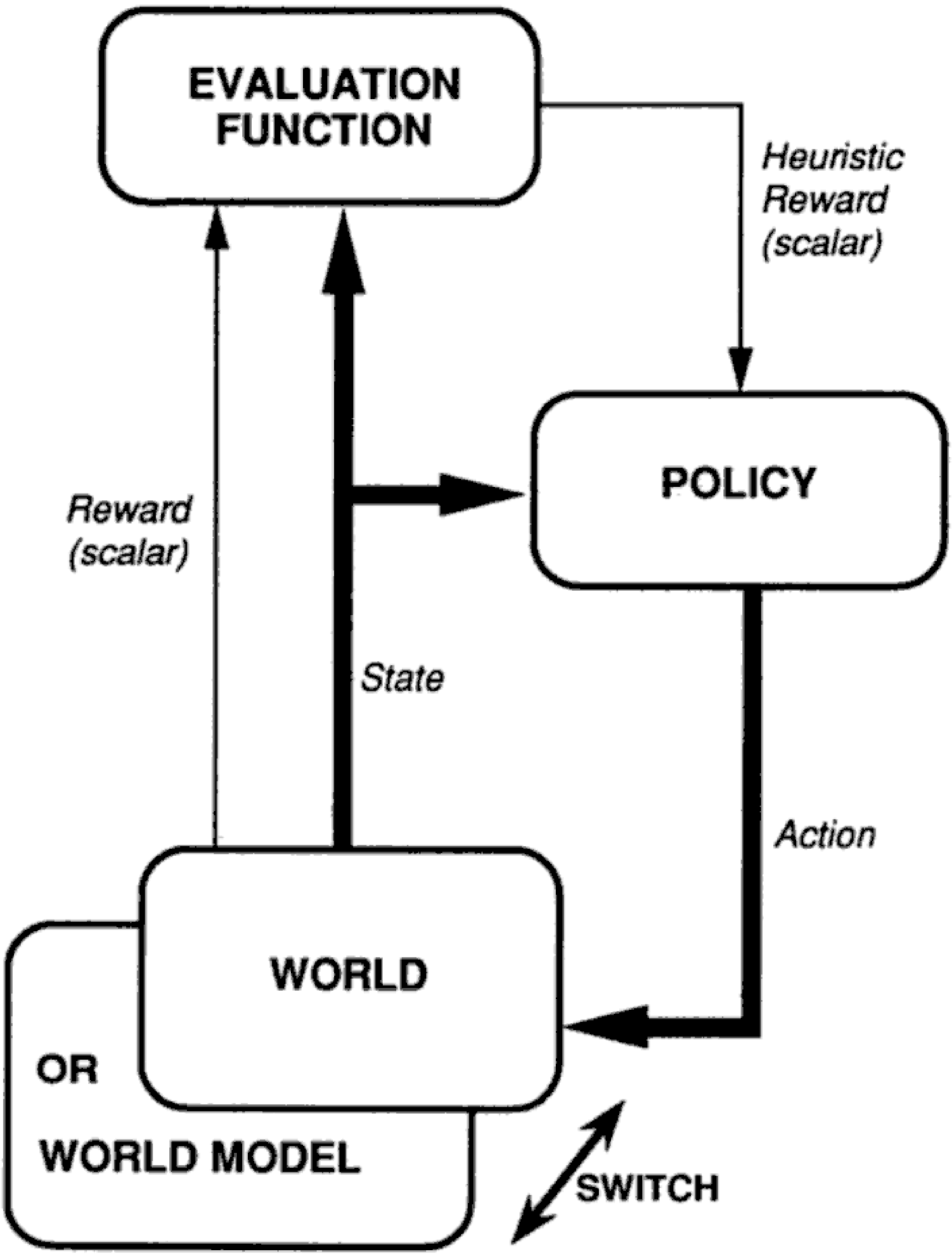

**Figure 8.1**
Overview of Dyna.

connectionist networks, or symbolic rules. Although the existing theory does not directly apply to the case in which these machine learning algorithms are used, it does provide a theoretical foundation for exploring their use. Finally, this kind of planning also extends conventional planning in that it is applicable to stochastic and uncertain worlds and to non-boolean goals.

The above discussion gives the general idea of relaxation planning, but not the exact form used in Dyna. Dyna is based on a closely related method known as *policy iteration* (Howard 1960), in which the evaluation function and policy are simultaneously approximated. In addition, Dyna is a *Monte Carlo* or *stochastic approximation* variant of policy iteration, in which the world model need only be sampled, not examined directly. Since the real world can also be sampled, by actually taking actions and observing the result, the world can be used in place of the world model in this method.

In this case, the result is not relaxation planning, but a trial-and-error learning process much like reinforcement learning (see Barto, Sutton, and Watkins 1989). In Dyna, both of these are done at once. The same algorithm is applied both to real experience (resulting in learning) and to hypothetical experience generated by the world model (resulting in relaxation planning). The results in both cases are accumulated in the policy and the evaluation function. There is insufficient room here to fully justify the algorithm, but it is quite simple and is given in outline form in figure 8.2.

As a simple illustration of the Dyna architecture, consider the navigation task shown in the upper right of figure 8.3. The space is a 6 by 9 grid of possible locations or states, one of which is marked as the starting state, "S", and one of which is marked as the goal state, "G". The shaded states act as barriers and cannot be entered. All the other states are distinct and completely distinguishable. From each 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 or outside the space, in which case the location is not changed. Reward is zero for all transitions except for those into the goal state, for which it is +1. Upon entering the goal state, the system is instantly transported back to the start state to begin the next trial. None of this structure and dynamics is known to the Dyna system a priori.

In this demonstration, the world was assumed to be deterministic, that is, to be a finite-state automaton, and the world model was implemented simply as next-state and reward tables that were filled in

```
0. Decide if this is a real experience or a hypothetical one.
1. Pick a state $x$. If this is a real experience, use the current state.
2. Form prior evaluation of $x$: $e \leftarrow Eval(x)$
3. Choose an action: $a \leftarrow Policy(x)$
4. Do action $a$; obtain next state $y$ and reward $r$ from world or world
   model.
5. If this is a real experience, update world model from $x$, $a$, $y$, and $r$.
6. Form posterior evaluation of $x$: $e' \leftarrow r + \gamma Eval(y)$
7. Update evaluation function so that $Eval(x)$ is more like $e'$ rather
   than $e$; this typically involves temporal-difference learning.
8. Update policy—strengthen or weaken the tendency to perform ac-
   tion $a$ in state $x$ according to $e' - e$.
9. Go to Step 0.
```

**Figure 8.2**
Inner loop of a Dyna algorithm. These steps are repeated continually, sometimes with real experiences, sometimes with hypothetical ones.

whenever a new state-action pair was experienced (Step 5 of figure 8.2). The evaluation function was also implemented as a table and was updated (Step 7) according to the simplest temporal-difference learning method: $Eval(x) \leftarrow Eval(x) + \beta(e' - e)$. The policy was implemented as a table with an entry $w_{xa}$ for every pair of state $x$ and action $a$. Actions were selected (Step 3) stochastically according to a Boltzmann distribution: $P\{a \mid x\} = e^{w_{xa}} / \sum_j e^{w_{xj}}$. The policy was updated (Step 8) according to: $w_{xa} \leftarrow w_{xa} + \alpha(e' - e)$. For hypothetical experiences, states were selected (Step 1) at random uniformly over all states previously encountered. The initial values of the evaluation function $Eval(x)$ and the policy table entries $w_{xa}$ were all zero; the initial policy was thus a random walk. The world model was initially empty; if a state and action were selected for a hypothetical experience that had never been experienced in reality, then the following steps (Steps 4–8) were simply omitted.

In this instance of the Dyna architecture, the inner loop (figure 8.2) was applied alternately to the real world and to the world model. For each experience with the real world, $k$ hypothetical experiences were

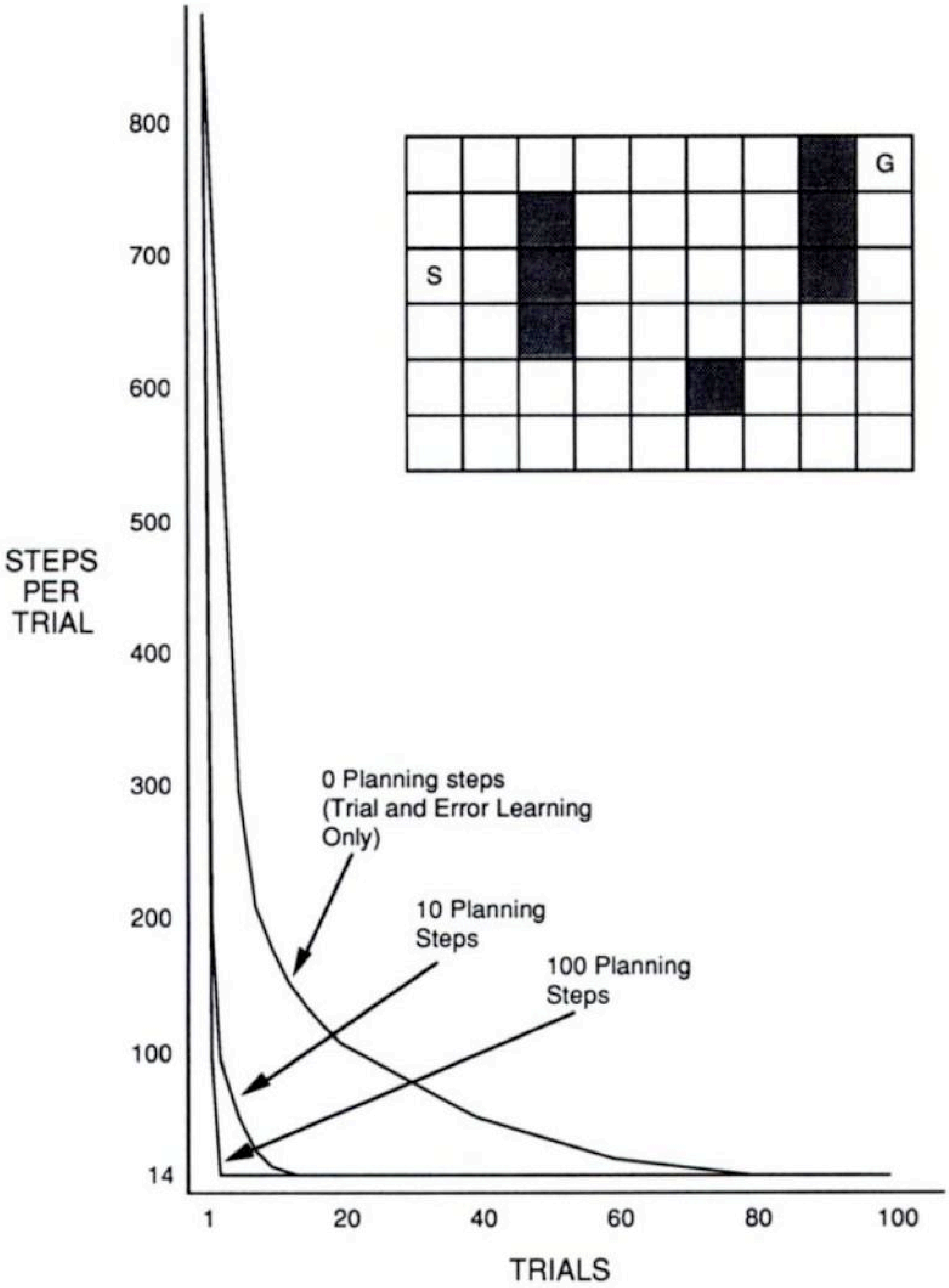

**Figure 8.3**
Learning curves for Dyna systems on a simple navigation task. A trial is one trip from the start state "S" to the goal state "G". The shortest possible trial is 14 steps. The more hypothetical experiences ("planning steps") using the world model, the faster an optimal path was found.

generated with the model (Step 0). Figure 8.3 shows learning curves for $k = 0$, $k = 10$, and $k = 100$, each an average over 100 runs. The $k = 0$ case involves no planning; this is a pure trial-and-error learning system entirely analogous to those used in reinforcement learning systems (Barto, Sutton, and Anderson 1983, Sutton 1984, Anderson 1987). Although the length of path taken from start to goal falls dramatically for this case, it falls much *more* rapidly for the cases including hypothetical experiences (planning), showing the benefit of using a learned world model. For $k = 100$, the optimal path was generally found and followed by the fourth trip from start to goal; this is very rapid learning. The parameter values used were $\beta = 0.1$, $\gamma = 0.9$, and $\alpha = 1000$ ($k = 0$) or $\alpha = 10$ ($k = 10$ and $k = 100$). The $\alpha$ values were chosen roughly to give the best performance for each $k$ value.

Figure 8.4 shows why a Dyna system that includes planning solves this problem so much faster than one that does not. Shown are the policies found by the $k = 0$ and $k = 100$ Dyna systems halfway 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 the last step, but here during the second trial an extensive policy has been developed that by the trial's end will reach almost back 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.

This simple illustration is 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. The Dyna architecture is fully compatible with the use of a wide range of learning methods for doing this. In this example, it was also assumed that the Dyna system has 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 and Schapire 1985, Mozer and Bachrach 1989). The Dyna architecture 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 and Chapman 1987, Whitehead 1989).

Yet another limitation of the example Dyna system presented here is the trivial form of search control used. Search control in Dyna boils down

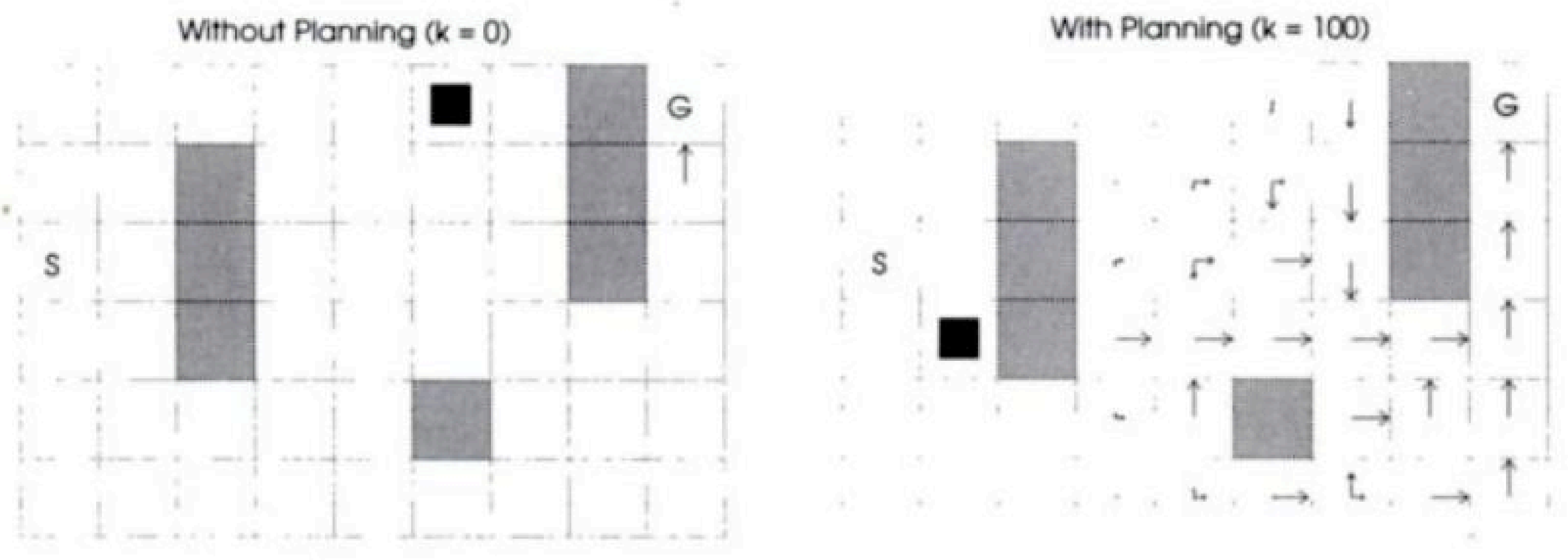

**Figure 8.4**
Policies found by planning and non-planning Dyna systems by the middle of the second trial. The black square indicates the current location of the Dyna system, and the arrows indicate action probabilities (excess over the smallest) for each direction of movement.

to the decision of whether to consider hypothetical or real experiences, and of picking the order in which to consider hypothetical experiences. The task considered here is so small that search control is unimportant, and was thus done trivially, but a wide variety of more sophisticated methods could be used. Particularly interesting is the possibility of using the Dyna architecture at a higher level to make these decisions.

Finally, the example presented here is limited in that reward is only nonzero 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. In the general case, the Dyna algorithm given here attempts to maximize the cumulative reward received per time step.

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 the functionality of planning can be obtained in a completely incremental manner, and how a planning process can be freely intermixed with reaction and learning processes. I conclude that it is not necessary to choose between planning systems, reactive systems, and learning systems. These three can be integrated not just into one system, but into a single algorithm, where each appears as a different facet or slightly different use of that algorithm.

## Acknowledgments

The author gratefully acknowledges the extensive contributions to the ideas presented of his colleagues Andrew Barto and Chris Watkins. I also wish to also thank the following people for ideas and discussions: Steve Whitehead, 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., and 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. Irvine, CA: Morgan Kaufmann.

Barto, A. G., Sutton R. S., and 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., and 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.

Bellman, R. E. (1957). *Dynamic Programming.* Princeton, NJ.: Princeton University Press.

Craik, K. J. W. (1943). *The Nature of Explanation.* Cambridge, UK.: Cambridge University Press.

Dennett, D. C. (1978). Why the law of effect will not go away. In *Brainstorms*, by D. C. Dennett, 71–89, Montgomery, Vermont: Bradford Books.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* New York: Wiley.

Mozer, M. C., and Bachrach, J. (1989). Discovering the structure of a reactive environment by exploration. Technical Report CU-CS-451-89, Dept. of Computer Science, University of Colorado at Boulder.

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

Ross, S. (1983). *Introduction to Stochastic Dynamic Programming*. New York: Academic Press.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton, G. E. (1986) Schemata and sequential thought processes in PDP models. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume II*, by J. L. McClelland, D. E. Rumelhart, and the PDP research group, 7–57.

Russell, S. J. (1989). Execution architectures and compilation. *Proceedings IJCAI-89*, 15–20.

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

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

Sutton, R. S., Barto, A. G. (1981). An adaptive network that constructs and uses an internal model of its environment. *Cognition and Brain Theory Quarterly 4*: 217–246.

Sutton, R.S., Pinette, B. (1985). The learning of world models by connectionist networks. *Proceedings of the Seventh Annual Conf. of the Cognitive Science Society*, 54–64.

Watkins, C. J. C. H. (1989). *Learning with Delayed Rewards*. PhD thesis, Cambridge University.

Werbos, P. J. (1987). Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, Jan-Feb.

Whitehead, S. D. (1989). Scaling reinforcement learning systems. Technical Report 305, Dept. of Computer Science, University of Rochester, Rochester, NY 14627.