Appeared in *Working Notes of the 1991 AAAI Spring Symposium*, pp. 151–155, and *SIGART Bulletin*, *2*, pp. 160-163, 1991

# Dyna, an Integrated Architecture for Learning, Planning, and Reacting

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

## Abstract

Dyna is an AI architecture that integrates learning, planning, and reactive execution. Learning methods are used in Dyna both for compiling planning results and for updating a model of the effects of the agent's actions on the world. Planning is incremental and can use the probabilistic and ofttimes incorrect world models generated by learning processes. Execution is fully reactive in the sense that no planning intervenes between perception and action. Dyna relies on machine learning methods for learning from examples—these are among the basic building blocks making up the architecture—yet is not tied to any particular method. This paper briefly introduces Dyna and discusses its strengths and weaknesses with respect to other architectures.

## 1 Introduction to Dyna

The Dyna architecture attempts to integrate

- Trial-and-error learning of an optimal *reactive policy*, a mapping from situations to actions;
- Learning of domain knowledge in the form of an *action model*, a black box that takes as input a situation and action and outputs a prediction of the immediate next situation;
- Planning: finding the optimal reactive policy given domain knowledge (the action model);
- Reactive execution: No planning intervenes between perceiving a situation and responding to it.

In addition, the Dyna architecture is specifically designed for the case in which the agent does not have complete and accurate knowledge of the effects of its actions on the world and in which those effects may be nondeterministic.

Dyna assumes the agent's task can be formulated as a *reward maximization* problem (Figure 1). At each discrete time interval, the agent observes a situation, takes an action based on it, and then, after one clock tick, observes a resultant reward and new situation. The agent's objective is to choose actions so as to maximize the total reward it receives in the long-term.[1] This problem formulation has been used in studies of reinforcement learning for many years and is also being used in studies of planning and reactive systems (e.g., Russell, 1989). Although somewhat unfamiliar, the reward maximization problem is easily mapped onto most problems of interest.

[1]Most systems actually slightly discount delayed reward relative to immediate reward.

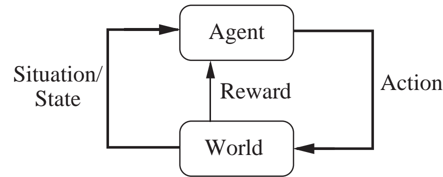


Figure 1: The Problem Formulation Used in Dyna. The agent's object is to maximize the total reward it receives over time.[1]

REPEAT FOREVER:

**1.** Observe the world's state and reactively choose an action based on it;

**2.** Observe resultant reward and new state;

**3.** Apply reinforcement learning to this experience;

**4.** Update action model based on this experience;

**5.** Repeat $K$ times:

**5.1** Choose a hypothetical world state and action;
**5.2** Predict resultant reward and new state using action model;
**5.3** Apply reinforcement learning to this hypothetical experience.

Figure 2: A Generic Dyna Algorithm.

The main idea of Dyna is the old, commonsense idea that planning is 'trying things in your head,' using an internal model of the world (Craik, 1943; Dennett, 1978; Sutton & Barto, 1981). This suggests the existence of a more primitive process for trying things *not* in your head, but through direct interaction with the world. *Reinforcement learning* is the name we use for this more primitive, direct kind of trying, and Dyna is the extension of reinforcement learning to include a learned world model.

The essence of Dyna is given by the generic algorithm in Figure 2. In this algorithm, an "experience" is a single unit of experience consisting of a starting state, an action, a resulting state, and a resulting reward. The first step of the algorithm is simply that of a reactive system; the agent reads off of its reactive *policy* what to do in the current situation. The first three steps together comprise a standard reinforcement learning agent. Given enough experience, such an agent can learn the optimal reactive mapping from situations to action. The fourth step is the learning of domain knowledge in the form of an *action model* (Lin, 1991) that can be used to predict the results of actions. The fifth step of the algorithm is essentially reinforcement learning from hypothetical, model-generated experiences; this is in effect a planning process.

The theory of Dyna is based on the theory of dynamic programming (e.g., Bertsekas, 1987) and on the relationship of dynamic programming to reinforcement learning (Watkins, 1989; Barto, Sutton & Watkins, 1990), to temporal-difference learning (Sutton, 1988), and to AI methods for planning and search (Korf, 1990). Werbos (1987) has previously argued for the general idea of building AI systems that approximate dynamic programming, and Whitehead (1989) and others have presented results for reinforcement learning systems augmented with with an action model used for planning. More recently, Riolo (1991) and Grefenstette et al. (1990) have explored in different ways the use of action models together with reinforcement learning methods based on classifier systems. Mahadevan and Connell (1990) have applied reinforcement learning methods together with ideas from subsumption architectures to a real robotic box-pushing task. Lin has explored Dyna architectures and related ideas on both simulated (Lin, 1991) and real robot tasks (Lin, personal communication).

## 2 Components of Dyna

Instantiating the Dyna architecture involves selecting three major components:

- The structure of the action model and its learning algorithms;
- An algorithm for selecting hypothetical states and actions (Step 5.1, search control).
- A reinforcement learning method, including a learning-from-examples algorithm and a way of generating variety in behavior.

The structure and learning of the action model lie mostly outside the the scope of the Dyna architecture. Recall that the action model is meant to be simply a mimic of the world; it takes in a description of a state and an action and emits a prediction of the immediate resulting state and reward. Actual experience with the world continually produces examples of desired behavior for such a model. These can be used in conjunction with any of a large number of learning algorithms for supervised learning (learning from examples). The design of that algorithm, its knowledge representation and generalization capabilities will of course have a large effect on the quality of the learned model, on how efficiently it is learned, and on how easily it can be primed with prior domain knowledge. Nevertheless, we consider those issues to be outside the scope of the Dyna architecture per se. Because Dyna makes no strong assumptions about the action model, it can use a wide variety of methods now existent or yet to be developed. One assumption Dyna does make that is not true of some supervised learning methods is that they can operate *incrementally*, that is, processing examples one by one rather than saving them up and making multiple passes.

At this time little can be said about how hypothetical starting states and actions should be selected. It can be done in a large variety of ways, but there has been little experience with any but the simplest. For example, in my previous work I have selected among previously observed states at random, either uniformly or in proportion to their frequency of prior occurrence. This is essentially the issue of *search control*—what part of the state space shall be worked on (planned about) next? Larger problems will of course require that the search be controlled more carefully. For some choices of search control method, the form of planning done in Dyna may be essentially the same as traditional kinds of planning, but for others it is clearly different. The following section discusses planning in Dyna further.

Among the reinforcement learning algorithms that can be used in Steps 3 and 5.3 of the Dyna algorithm (Figure 2) are the adaptive heuristic critic (Sutton, 1984), the bucket brigade (Holland, 1986), and other genetic algorithm methods (e.g., Grefenstette et al., 1990). For concreteness, consider the simplest, most recent, and perhaps most promising method, *Q-learning* (Watkins, 1989). The basic idea in Q-learning is to learn an evaluation function that gives the value of performing each action in each state. This function is usually denoted $Q(x,a)$, where $x$ is a state and $a$ is an action (the name "Q-learning" comes from this choice of notation). When using Q-learning, the action chosen in a state $x$ is usually simply the action $a$ for which $Q(x,a)$ is maximal.

The update algorithm for Q-learning can be expressed in a general form as a way of moving from a unit of experience to a training example for the evaluation function. This training example is then input to a supervised learning algorithm. Just as in learning the action model, the choice of supervised learning algorithm will have a strong effect on the performance of the Dyna architecture, but is not a part of the architecture itself. Recall that a unit of experience consists of a starting state ($x$), an action ($a$), a next state ($y$), and a reward ($r$). From this one forms the training example:

$$Q(x,a) \quad \text{should be} \quad r + \gamma \max_b Q(y,b),$$

where $\gamma$, $0 \leq \gamma < 1$, is a constant that determines the relative value of short-term versus long-term reward. Strong formal results are available for the case in which the $Q$ function is implemented as a table. For that case, Watkins (1989) has shown that Q-learning from real experiences—direct agent-environment interaction without using an action model—will converge to the optimal behavior under weak conditions.

## 3 Planning and Reacting in Dyna

Just as reinforcement learning with real experience (Steps 1–3) is meant to learn the optimal way of behaving for the real world, reinforcement learning with hypothetical experience (Steps 5.1–5.3) is meant to learn the optimal way of behaving given the action model. Reinforcement learning with hypothetical experience is in fact an incremental form of planning that is closely related to dynamic programming. Here we will call it *incremental dynamic programming*, after Watkins (1989), or *IDP planning* for short. Assuming IDP planning steps can be done relatively quickly and cheaply compared to real steps (i.e., $K >> 1$) and that the model is correct, IDP planning will greatly speed the finding of the optimal policy. In small tasks this has been shown to be true even if the model must be learned as well or if the world changes (Sutton, 1990).

Results from dynamic programming (Bertsekas & Tsitsiklis, 1989) can be adapted to show that IDP planning based on the tabular version of Q-learning converges onto the optimal behavior given the action model. This is a strong result because it applies to nondeterministic environments and no matter how deep a search is required to find the optimal actions. Strictly, it applies only to the tabular case, but the results should be similar for supervised learning methods to the extent that they can accurately approximate the desired functions.

Dyna is *fully reactive* in the sense that *no* planning intervenes between observing a state and taking an action dependent

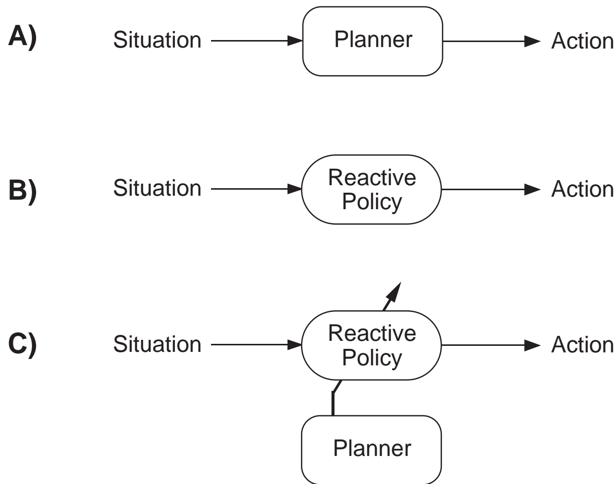


Figure 3: Simplistic Comparison of Architectures: A) Conventional Planning, B) Reactive Systems, C) IDP Planning (incremental compiling into reactions).

on that state. In the Dyna algorithm given in Figure 2, IDP planning takes place after action selection, but conceptually these processes proceed in parallel.[2] The critical issue is that planning and reacting processes are not strongly coupled: the agent never delays responding to a situation in order to plan a response to it. Although the agent always responds reactively and instantly, this does not mean it must immediately respond decisively; for example, it may choose the response of sitting still. Figure 3 contrasts this approach to combining planning and execution with that of conventional planning systems and of reactive systems.

IDP planning has both advantages and disadvantages compared to other planning methods. The primary advantage is that it is totally incremental; any time spent planning results in an improvement in the agent's immediate reactions or evaluation function for some state. Thus, performance continually improves, and arbitrarily long optimal sequences of actions can be found. In addition, it readily handles nondeterministic tasks and is extremely general in that it makes no assumptions about the world other than that is can be at least partially predicted.

The primary disadvantage of IDP planning is that it may require large amounts of memory. Whereas traditional planning methods are based on constructing search trees and backing-up evaluations on demand, IDP planning is based on storing backed-up evaluations (and possibly reactions) associated with each state or state-action pair. Even if supervised learning methods are used instead of tables, this is still a memory-intensive approach. It will require far more memory than depth-first search, for example.

## 4 Potential Problems with Dyna

In the rest of this paper we briefly discuss a number of potential problems with the Dyna architecture.

[2]The Dyna algorithm given in Figure 2 also sacrifices reactivity somewhat for the sake of pedagogy. A more fully reactive version of the algorithm would move Step 5 inbetween Steps 1 and 2. More generally, the four main functions of the algorithm—reacting, reinforcement learning, model learning, and IDP planning—should be thought of as running simultaneously and independently.

### 4.1 Reliance on Supervised Learning

On realistic problems, the state space is obviously far too large for table-based approaches, and thus Dyna must rely on methods for learning and generalizing from examples. However, despite enormous amounts of work in several disciplines, fully satisfactory methods for supervised learning remain to be found. For example, there remain difficult issues in generalization and knowledge representation that must be solved. Nevertheless, I do not feel it is inappropriate to base an integrated architecture on a capability for effectively learning from examples. Would not any integrated architecture rely on such a capability at some level? Any architecture using analogy, compilation, reminding, or even similarity would do so. If the answer is clearly 'yes,' then why not build this in as a basic part of the architecture?

### 4.2 Hierarchical Planning

Dyna as described is a very flat system. It plans at the level of individual actions. If those actions are muscle twitches, then Dyna will be of no help planning a trip across the country—and neither will any other planner that operates at a single level. Planning must be done at several levels and the results combined in some way. We have had lots of experience doing this with conventional planners, but it has not been tried with Dyna. To my knowledge there is no reason as yet to think that hierarchical planning will be either easier or harder in Dyna than it is in conventional planners.

### 4.3 Ambiguous and Hidden State

So far we have assumed that the agent can observe the relevant aspects of the world's state at no cost and on every time step, assumptions that are clearly violated in many tasks of interest. This is a limitation that Dyna shares with most other planning and problem solving systems—they are all based on state. For example, a robot may not be able to determine from its immediate surroundings which of two similar rooms it is in, or whether a door is locked, or whether there is a person in the room on the other side of the door. In these cases the robot cannot unambiguously determine the world's state, as much of it is hidden from him.

There are a number of techniques for dealing with this problem, though none is clearly a general solution. In some cases, uncertainty about the true state on the world can be modeled as probabilistic state transitions (Kaelbling, 1990). Approaches such as Dyna that can handle stochastic tasks can then be used without change. In other cases, the state description can be augmented with past inputs to disambiguate state. For example, a robot may not be able to sense a wall in front of it, but if it remembers that it just bumped into it and backed off, and makes that memory part of the current state description, then the situation can be handled by state-based methods.

Whitehead and Ballard (1991) have proposed learning perceptual strategies for disambiguating state descriptions created by a marker-based visual system. Ming Tan (personal communication) has also explored the use of Cost-Sensitive learning in reinforcement learning for a similar purpose. There is considerable relevant work in the dynamic programming literature, but that direction has not been explored yet.

### 4.4 Ensuring Variety in Behavior

In order to maintain an accurate action model, the agent must try actions that it believes to be inferior. If it only tries those that it believes are best, and the world changes, it may never discover the change and never discover whatever new actions are really best. The simplest way to ensure behavioral variety is to require the agent to choose an action at random a small percentage of the time. This crude strategy has many disadvantages, but is adequate for many problems. Another approach is to choose actions based on a probability distribution, such as a Boltzmann distribution, that favors the apparently best actions, but does not select them 100% of the time. If desired, the 'temperature' of the distribution can be reduced over time to increase the preference for the apparent best actions (Watkins, 1989), but this creates again the inability to handle long-term changes in the world. The 'adaptive heuristic critic' architecture (Sutton, 1984) also has this problem. Perhaps the best solution developed so far, though still far from perfect, is the exploration bonus proposed by Sutton (1990).

### 4.5 Taskability

Superficially, the Dyna architecture is not taskable. Dyna is based on the reward maximization problem (Figure 1) which recognizes only one goal, the maximization of total reward over time. In addition, the object of the planning and learning processes are to learn one policy function that maps states to actions with no explicit 'goal' input. However, this may merely mean that the goal specification must be part of the state description. For example, consider a Dyna robot rewarded for picking up trash, but which must recharge its battery occassionally. When its battery is running low the optimal behavior will be to search out the recharger, whereas when it has plenty of power the optimal behavior will be to search out more trash. If the charge on the battery is part of the state description then these two apparent goals can easily be part of a single policy.

Similarly, to train a dog, e.g., to heel or to roll over, one provides distinctive cues, e.g., movements or sounds, that signal to the animal which of its actions will be rewarded *now*. It can be time-consuming to teach animals new behaviors because of the absence of a common language. It may be possible to task Dyna agents more directly than that. If one directly modifies the part of the action model that predicts reward, that could in turn cause the policy to change substantially through IDP planning.

### 4.6 Incorporation of Prior Knowledge

Prior knowledge can be incorporated in Dyna systems through the initial values of the policy and internal evaluation functions such as the $Q$ function. In principle this could be a very flexible and efficient method, but there is little work with it yet. Lin (personal communication) has demonstrated in preliminary results a very effective method that he calls 'teaching' in which an outside agent, say a human expert, takes control over the agent and demonstrates a correct solution to the problem. This experience is processed by the Dyna system (or, in Lin's case, Dyna-like system) in the normal way, and greatly speeds subsequent learning.

## References


Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1990) Learning and sequential decision making. In *Learning and Computational Neuroscience,* M. Gabriel and J.W. Moore (Eds.), 539–602, MIT Press.

Bertsekas, D. P. (1987) *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall.

Bertsekas, D. P. & Tsitsiklis, J. N. (1989) *Parallel Distributed Processing: Numerical Methods*, Prentice-Hall.

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

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

Grefenstette, J. J., Ramsey, C. L., & Schultz, A. C. (1990) Learning sequential decision rules using simulation models and competition. *Machine Learning 5*, 355–382.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. Michalski, J. Carbonell & T. Mitchell, Eds., *Machine learning II*, Morgan Kaufmann.

Kaelbling, L. P. (1990) *Learning in Embedded Systems.* Ph.D. thesis, Stanford University.

Korf, R. E. (1990) Real-Time Heuristic Search. *Artificial Intelligence 42*: 189–211.

Lin, Long-Ji. (1991) Self-improving reactive agents: Case studies of reinforcement learning frameworks. In: *Proceedings of the International Conference on the Simulation of Adaptive Behavior*, 297–305, MIT Press.

Mahadevan, S. & Connell, J. (1990) Automatic programming of behavior-based robots using reinforcement learning. IBM technical report.

Riolo, R. (1991) Lookahead planning and latent learning in a classifier system. In: *Proceedings of the International Conference on the Simulation of Adaptive Behavior*, MIT Press.

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

Sutton, R. S. (1984) Temporal credit assignment in reinforcement learning. PhD thesis, COINS Dept., Univ. of Mass., 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.

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.

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

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*, SMC-17, No. 1, 7–20.

Whitehead, S. D., Ballard, D.H. (1991) Learning to perceive and act by trial and error. *Machine Learning 7:*, 45-83.

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