Appeared in *Proceedings Ninth Conference on Machine Learning*, 353–357, 1991 Morgan-Kaufmann

Planning by Incremental Dynamic Programming

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

Abstract

This paper presents the basic results and ideas of dynamic programming as they relate most directly to the concerns of planning in AI. These form the theoretical basis for the incremental planning methods used in the integrated architecture *Dyna*. These incremental planning methods are based on continually updating an evaluation function and the situation-action mapping of a reactive system. Actions are generated by the reactive system and thus involve minimal delay, while the incremental planning process guarantees that the actions and evaluation function will eventually be optimal—no matter how extensive a search is required. These methods are well suited to stochastic tasks and to tasks in which a complete and accurate model is not available. For tasks too large to implement the situation-action mapping as a table, supervised-learning methods must be used, and their capabilities remain a significant limitation of the approach.

1 INTRODUCTION

Planning has classically been viewed as something that *intervenes* between situation and action: as each situation is encountered, an appropriate action in response to it is planned, and only when planning is complete is an action actually taken. However, this approach creates a basic problem—any time spent planning is time by which the action is delayed. It has appeared at times that response could be fast only if planning was shallow and weak.

Faced with this dilemma, several researchers have explored the radical idea that planning might not play a central role in decision making. Agre and Chapman (1987; Agre, 1988) proposed that almost all human behavior is routine, that most of the time people simply know what to do without planning. They and others (Schoppers, 1987; Rosenschein and Kaelbling, 1986) developed the notion of a *reactive system*, a system in which no planning intervenes between situation and action, but instead the action taken is computed rapidly as a function of the situation. Other work has attempted to take an intermediate ground and manage the tradeoff flexibly at runtime (e.g., Dean & Boddy, 1988)

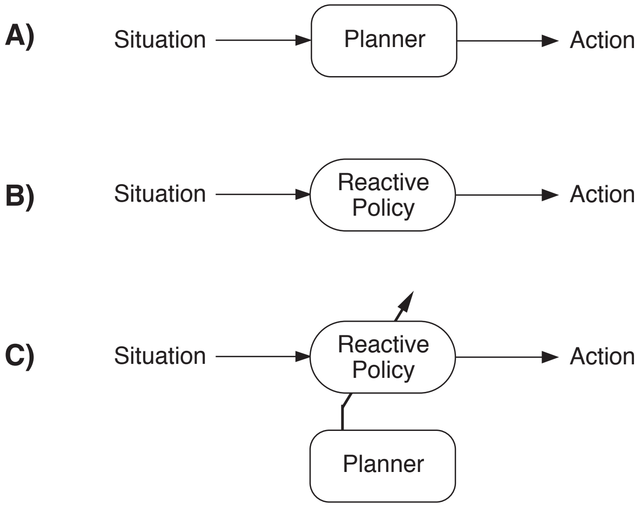

Figure 1: Simplistic Comparison of Architectures: A) Conventional Planning, B) Reactive Systems, C) Planning Compiled into Reactions (the approach taken in Dyna).

The *Dyna* integrated architecture (Sutton, 1990) combines reactive systems and planning in a simple way. Actions are generated by a reactive system, but the mapping implemented by that system is continually adjusted by a planning process, as suggested by Figure 1c. Planning and reaction proceed independently and conceptually in parallel; the reactive system never waits for the planner and the planner is not restricted to planning about the current situation.

This approach to planning can be viewed as an application and extension of *dynamic programming* (DP). DP is a broad class of techniques developed in operations research for solving sequential decision problems such as routing and scheduling (e.g., Bertsekas, 1987). These methods are in many ways very similar to AI search methods, but they differ in that they allocate memory to the entire state space instead of only to a tree of possibilities starting at the current state. Incremental DP methods improve over conventional AI planners in that they are readily applicable to stochastic problems and in that they are incremental.

Incremental DP methods can be viewed as similar to "speed-up learning" or "plan-then-compile" architectures (e.g., Mitchell, 1990; Laird & Rosenbloom, 1990). In a sense, incremental DP methods represent this idea carried to an extreme: all planning is done by compiling minimal (one-ply) searches; there is no other planning process.

In this paper we present the basic ideas of dynamic programming and incremental dynamic programming as they relate most directly to the concerns of planning in AI. We do not present any completely new results, but rather we present existing results in a way that is more complete and more focussed on planning than has been done previously. The original sources of the ideas and results include Bellman (1957), Samuel (1959), Holland (1986), Werbos (1987), Watkins (1989), Bertsekas & Tsitsiklis (1989), Korf (1990), Barto et al. (1990), Whitehead (1989; Whitehead & Ballard, in press), and Sutton (1988, 1990). The presentation here is in two sections. The first section treats a conceptually very simple case that should be familiar to AI practitioners; here the basic idea and results of incremental dynamic programming as a fully incremental form of planning are introduced. In the second section these ideas are generalized and extended to the stochastic case and to a class of problems more like those likely to arise in practice.

In general, strong formal results can be obtained only for the case in which each world state is recognized individually and the policy and evaluation functions are implemented as tables. This is the only case treated in this paper. Dyna-like systems that go beyond the tabular case have been investigated by Lin (1991), Moore (1990), Riolo (1991), and Watkins (1989).

2 THE COST-TO-GOAL TASK

To begin, let us consider a particularly simple case. At each of a sequence of discrete time steps, $t = 0, 1, 2, \ldots$, a problem-solving agent observes its world to be in a state $s \in S$ and then sends an action $a \in A_s$ to the world (Figure 2). Each action incurs a cost $c_{sa} \in \Re^+$, and the agent's objective is to select actions so as to drive the world to a goal state $g \in G \subset S$ with minimum cumulative cost. For the moment we assume that the world is deterministic and that it is possible to reach the goal from all states.

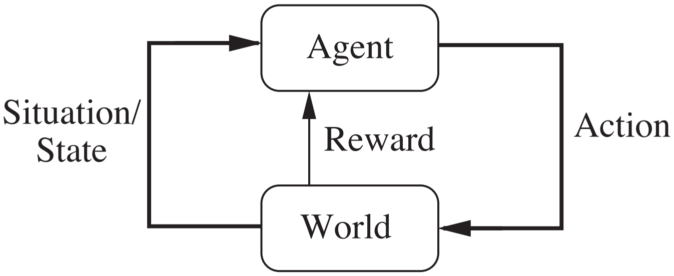

Figure 2: The Agent–Environment Interaction. The agent's object is to minimize (maximize) its cumulative cost (reward) over time.

Heuristic search and dynamic programming are both based on assigning a numerical *value*, $V(s)$, to each state. In this problem the natural measure of value is the minimum cost-to-go from that state to a goal state. These values can be uniquely defined by a recursive relationship: the minimal cost-to-go for a state must equal the cost for the best action plus the minimal cost-to-go for the state that results:

$$V(s) = \min_{a \in A_s} \quad c_{sa} + V(succ(s,a)) \qquad \forall s \in S - G \quad (1)$$

where $succ(s,a)$ denotes the successor state to s given the selection of action a, and where

$$V(s) = 0. \qquad \forall s \in G \qquad (2)$$

The equation (2) defines the values of the goal states, which form the terminal nodes of a search tree, and (1) specifies how these are *backed up* to define the values of all other states. Once the values of the immediate successors to the root state s are known, the best action is that which minimizes the righthand side of (1).

In practice the computation specified by (1) and (2) is far too complex to carry out on any but the smallest of problems. It involves backing up every interior node in the tree, from the starting state to the terminals. To avoid this, most conventional search procedures approximate this computation by searching only a subtree—for example, by searching only to a limited depth. At the frontier of this subtree the values are estimated heuristically using an approximate value function $h(s) \approx V(s)$. These approximate values are then backed up just as the true ones were before. That is, (1) is replaced by

$$\hat{V}(s) = \min_{a \in A_s} \quad c_{sa} + \hat{V}(succ(s,a)), \qquad (3)$$

for all states s on the interior of the subtree, and (2) is replaced by

$$\hat{V}(s) = h(s), \qquad (4)$$

for all states s on the frontier.

Although search processes like (3) and (4) are very widely used, they are very wasteful if the same or similar search problems recur. The whole point of the search is to propagate approximate values h from the leaf nodes up to produce better approximate values $\hat{V}$, but after each search the backed-up values are simply discarded. If instead these were retained as the heuristic value function for the next search, then that search would in effect continue from where the previous one left off. In this case one would have not two approximate value functions, $\hat{V}$ and h, but only one, $\hat{V}$, which is continually improved by backing up its own values. There are a number of variations on this idea.

The simplest case is that in which the state space is finite and the approximate value of each state can be accessed individually, as if it was an entry in a large table. In this *tabular* case, the natural operation for improving the approximate value function $\hat{V}(s)$ at a state $s \in S - G$ is simply (3) converted from an equation to an assignment:

$$\hat{V}(s) := \min_{a \in A_s} \quad c_{sa} + \hat{V}(succ(s,a)), \qquad (5)$$

where, for $s \in G$, $\hat{V}(s) = V(s) = 0$. This operator is essentially a one-ply search from state s followed by the replacement of the approximate value $\hat{V}(s)$ with its backed-up value.[1]

The central idea explored in this paper is that of repeatedly applying improvement operators like (5) to one state after another such that $\hat{V}$ becomes a better and better approximation to V. First note that if this process stops, if (5) no longer causes the value of any state to change, then (1) holds for $\hat{V}$, and therefore $\hat{V} = V$. Furthermore, for the tabular case, it has been proven that $\hat{V}$ converges to V given only that all states continue to be updated by (5) (Bertsekas & Tsitsiklis, 1989).

Following Bertsekas and Tsitsiklis, we call this way of computing V *asynchronous value iteration (AVI)*, because it is essentially an asynchronous version of a classic dynamic programming method known as *value iteration*. In conventional value iteration, the improvement operator is applied not to one state after another, but to all of them simultaneously, in one big, synchronized, parallel step. That step is then repeated until convergence.

From the point of view of AI, AVI is simply an incremental search or planning method. For example, search control in AVI arises as the issue of which states should be updated with what frequencies. The convergence result requires only that all states be visited occasionally, leaving wide latitude for the use of heuristics to control the search.

In practice, the state spaces are often too large to implement $\hat{V}$ as a table or to visit all states over and over again as are required for this result. Nevertheless, the result shows that if there were time and space enough, then convergence would occur onto the optimal value function and the optimal behavior. Note that this is not true of ordinary heuristic search, which produces an approximate result whenever the search tree cannot be exhaustively searched during *one response time*. Incremental planning methods based on (5) are essentially a way of trading memory for search. By devoting memory to the approximation $\hat{V}$ they accumulate individual searches into an overall, continuously improving result that can be far better than any feasible single search. The advantages of this strategy become most significant for non-deterministic tasks.

3 STOCHASTIC AND CONTINUING TASKS

The previous section presented the basic ideas of relaxation planning in terms of a simple special case. In this section, we generalize those ideas and present new procedures and results for a more general and broadly applicable case. In this section we allow the effects of the agent's actions on the world to be probabilistic as well as deterministic, and we allow for a more general class of agent objectives.

The cost-to-goal problem developed in the previous section is delightfully simple, but limited. One difficulty is that the notion of a goal state is too absolute and binary for many problems that involve more than one objective. Multiple objectives can sometimes be incorporated into per-step costs, but then they must be in a special, secondary relationship to the primary objective of reaching a goal state. A second difficulty with the cost-to-goal formulation is that it explicitly ignores what happens after reaching a goal state. In general, this is satisfactory only if what happens is independent of which goal state was reached. If the overall problem does not divide up naturally into independent problem-solving episodes (such as repeated plays of a game), then it can be difficult to use the cost-to-goal formulation.

Both of these difficulties with the cost-to-goal formulation are due to its use of special goal states that are different from ordinary states. One might well ask why special goal states are necessary given that the formulation already includes costs on state transitions. Why not make the objective simply the optimization of these costs, and do without goal states? We next propose a new problem formulation based on this idea.

First, we generalize the idea of per-transition costs, to per-transition outcomes, called *rewards*. Rewards can be either desirable (positive) or undesirable (negative), where the latter are essentially the same as costs. The

[1] Similar operators could be defined for larger, deeper searches, but for the tabular case that is probably not better than multiple one-ply searches.

agent's objective is to choose actions so as to maximize the total reward received from the world (cf. Russell, 1989). For example, a robot meant to find and collect soda cans might be given a positive reward for depositing a can in the recycling bin, a negative reward for colliding with anything or if its battery runs too low, and zero reward at all other times. In a navigation problem, positive reward might be given upon arriving at the goal location, and perhaps slightly negative reward on transitions to states other than the goal, to discourage longer than necessary paths. In a chess-playing problem, reward might be made +1 for a win, -1 for a loss, and 0 for a draw and for all non-terminal positions.

At each of a sequence of discrete time steps, $t = 0, 1, 2, \ldots$, a problem-solving agent observes its world to be in a state $s_t \in S$ and sends an action $a_t \in A_s$ to the world (Figure 2). After each action, the agent receives from the world a reward $r_{t+1} \in \Re$ and the next state, s_{t+1}. The expected value of r_{t+1} is $R(s_t, a_t)$, and the probability that $s_{t+1} = x$, for any $x \in S$, is $P(s_t, x, a_t)$.

Informally, the agent's objective is to maximize the total reward it receives from the world. Intuitively, we would like to say that each action a_t is chosen to maximize the sum of all future reward, $r_{t+1}+r_{t+2}+r_{t+3}+\cdots$. However, because there are no goal states to terminate this sum, it is likely to be infinite or divergent. One way of solving this problem is to introduce discounting of rewards. The basic idea is that a unit of reward, like a unit of money, is worth more now than it will be later. Whereas a unit of reward arriving immediately has full value, a unit of reward arriving one time step later is discounted and worth only γ, $0 < \gamma < 1$, a unit arriving two steps later is discounted twice and worth only γ^2, and so on. In selecting a_t, the agent is then concerned with the *discounted* sum of future reward, $r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots$. This quantity is called the *return*. The agent's objective is to choose each action a_t so as to maximize the expected return:[2]

$$E\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right\}. \tag{7}$$

On navigation problems, discounting has the effect of encouraging short paths without an explicit cost for each step, as shorter paths will mean a less delayed, and therefore less discounted, terminal reward. In game-playing, discounting causes the agent to seek quick wins and slow losses. The discounted formulation is widely used in studies of sequential decision making and reinforcement learning, in part because it is mathematically convenient, and in part because it realistically represents the true goals in many tasks.

The *value* of a state, $V(s)$, can be defined as the best expected return attainable starting from that state, i.e., the expected return given optimal behavior. These values satisfy the recursive relationship

$$V(s) = \max_{a \in A_s} \; R(s,a) + \gamma V(succ(s,a)), \qquad \forall s \in S \tag{8}$$

where $succ(s, a)$ denotes the deterministic successor state to s given the selection of action a. Note the similarity of this relationship to (1) for the cost-to-goal problem. For the general case of probabilistic state transitions, the key recursive relationship is

$$V(s) = \max_{a \in A_s} \; R(s,a) + \gamma \sum_{x \in S} P(s,x,a) V(x). \qquad \forall s \in S \tag{9}$$

As in the cost-to-goal task, once the values of all the states are known, it is straightforward to determine the optimal actions. For any state s, the optimal action is that which attains the maximum in (9).

As in the cost-to-goal task, the improvement operator for an arbitrary $\hat{V} \approx V$ can be defined by turning the equation expressing the recursive relationship (9) into an assignment:

$$\hat{V}(s) := \max_{a \in A_s} \; R(s,a) + \gamma \sum_{x \in S} P(s,x,a) \hat{V}(x). \qquad \forall s \in S \tag{11}$$

As in the cost-to-goal task, if this operator is repeated applied to all states, in any order in which none are permanently left out, then $\hat{V}$ will converge to V (Bertsekas & Tsitsiklis, 1989; Watkins, 1989).

4 CONCLUSION

No completely new results have been presented in this paper, but rather we have collected and analyzed existing results from DP as a novel approach to incremental planning. For the tabular case, these results assure convergence onto the optimal behavior of a flexible incremental planning technique for general, stochastic problems. These results form the theoretical foundation for the Dyna architecture, and have not previously been presented from this perspective. Moreover, in this paper we have focussed on DP as the theoretical foundation for *planning* methods rather than as the theoretical foundation for reinforcement learning methods (see Watkins, 1989; Barto et al., 1990).

Incremental planning is advantageous in problem-solving tasks in which the same or similar problems recur. When encountering a familiar problem, an agent would like to take advantage of previous search, not start all over again. This is particularly important when fast response is required, as then little or no search can be done at the time. It should also be important whenever the agent has a changing model of the world, as that can be expected to give rise to a repetitive sequence of similar problems.

[2] It remains to show that there is such a way of choosing actions that maximizes the return at all states; this is shown by, e.g., Bertsekas (1987).

One lesson that can be taken from the work of Agre and Chapman (1987; Agre, 1988) is an emphasis on the repetitive nature of problem solving in everyday life. Most of the time people confront familiar problems and apply familiar solutions. Rarely do people explicitly stop and plan out their behavior. This argues against "one-shot" planning of the sort classically studied in AI, but not against incremental planning. It may not be that people rarely problem-solve, but rather that they problem-solve a little bit all the time—never starting from scratch but always making full use of their prior problem-solving experience.

Acknowledgements

The author gratefully thanks Andy Barto, Ron Williams, Steve Whitehead and Chris Watkins for numerous discussions and insights that have strengthened and clarified the presentation in this paper. I also thank Bernard Silver, Glenn Iba and the reviewers for constructive comments on an earlier draft.

References

Agre, P. E. (1988) *The Dynamic Structure of Everyday Life.* PhD thesis, MIT Technical Report AI-TR-1085.

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

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

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

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.

Dean, T., & Boddy, M. (1988) An analysis of time-dependent planning. *Proceedings AAAI-88*, 49–54.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach, Volume II*, Los Altos, CA: Morgan Kaufmann.

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

Laird, J. E., & Rosenbloom, P. S. (1990) Integrating planning, execution, and learning in Soar for external environments. AAAI-90, 1022–1029.

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

Mitchell, T. M. (1990) Becoming increasingly reactive. AAAI-90, 1051–1058.

Moore, A. W. (1990) *Efficient Memory-based Learning for Robot Control.* PhD thesis, Cambridge University Computer Science Department.

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.

Rosenschein, S. J., & Kaelbling, L. P. (1986) The synthesis of machines with provable epistemic properties. *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, 83–98, Morgan Kaufmann.

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

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, *3*, 210–229. Reprinted in E. A. Feigenbaum, & J. Feldman (Eds.), *Computers and thought.* New York: McGraw-Hill.

Schoppers, M. J. (1987) Universal plans for reactive robots in unpredictable domains. *Proceedings of IJCAI-87.*

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.

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*, Jan-Feb.

Whitehead, S. D., Ballard, D.H. (1990) Active perception and reinforcement learning. In: *Proceedings of the Sixth International Workshop on Machine Learning*, 354–357, Morgan Kaufmann.

Whitehead, S. D., Ballard, D.H. (in press) Learning to perceive and act by trial and error. *Machine Learning.*