Control Strategies for a Stochastic Planner *

Jonathan Tash
Group in Logic and the Methodology of Science
University of California
Berkeley, CA 94720
tash@math.berkeley.edu

Stuart Russell
Computer Science Division
University of California
Berkeley, CA 94720
russell@cs.berkeley.edu

Abstract

We present new algorithms for local planning over Markov decision processes. The *base-level* algorithm possesses several interesting features for control of computation, based on selecting computations according to their expected benefit to decision quality. The algorithms are shown to expand the agent's knowledge where the world warrants it, with appropriate responsiveness to time pressure and randomness. We then develop an *introspective* algorithm, using an internal representation of what computational work has already been done. This strategy extends the agent's knowledge base where warranted by the agent's world model and the agent's knowledge of the work already put into various parts of this model. It also enables the agent to act so as to take advantage of the computational savings inherent in staying in known parts of the state space. The control flexibility provided by this strategy, by incorporating natural problem-solving methods, directs computational effort towards where it's needed better than previous approaches, providing greater hopes for scalability to large domains.

Introduction

Planning under uncertainty is a good domain for investigating strategies for controlling the computational expenditures of a planner, because decision theory provides a language for expressing the sorts of tradeoffs between plan quality and solution time that a good control strategy should capture. This paper presents a planning methodology suitable for domains representable as a Markov decision process (as in (Koenig 1992)), and shows how such a methodology can incorporate several natural control strategies.

In this model, the problem-solving agent is in a world consisting of a finite, discrete set of states, and can identify the current one. In any given state, the agent has available to it a set of actions it can take, and for a given action choice, it knows the transition probabilities to other states. In addition, it has some reward function giving the immediate value of being in any particular state. For example, an agent attempting to reach a goal state as quickly as possible might assign the goal state a reward of 0 and all other states a reward of -1. Problems using such a reward function include the path-planning problem on a grid with obstacles and imperfect motor control, and the ubiquitous 8-puzzle, but with random errors associated with actions. (The model can also handle problems having several stop states of different values.) In this domain, a plan takes the form of a policy assigning to each state an action choice. The agent tries to choose a policy maximizing its cumulative reward. (For domains involving unbounded time, it is common to discount future gains by an amount exponential in time to avoid infinite utilities.)

The model is enriched by the inclusion of a penalty for deliberation time. The agent alternates between computing better policies and acting, and its cumulative reward is reduced as time is spent computing. This enrichment requires modification of the classical methods of operations research when faced with large domains.

The next section elaborates a planning system which accounts for the expense of computing by making tradeoffs between computation time and policy quality. The system uses local computation techniques resembling ones presented in (Dean *et al.* 1993) and (Thiébaux *et al.* 1994), but allows for more extensive use of heuristic knowledge and is amenable to a proof of convergence to optimality. In addition, our analysis provides a well-founded strategy for making decisions relevant to the control of the expensive computations, eliminating reliance on trial-and-error determination of good control procedures. This strategy is shown to have several interesting properties, such as providing appropriate responsiveness to increased time pressure and to increased randomness in the world.

We then discuss an extension to this system which allows for more refined control of the computational effort expended by the planner. This new system adds to the planner a representation of the computational effort it has already spent on various parts of the problem, enabling the planner to apply its knowledge of what it already knows about to its considerations of the value of its computational activities. Such a planner's control strategy has several new properties, counterparts to the more introspective aspects of natural problem-solving activity. These include the ability to direct its computational efforts according to where they

*This research was supported by NASA JPL, and by NSF awards IRI-9058427 (PYI) & IRI-9211512.

will have the most impact on current considerations, taking into account not only the planner's model of the world but its model of its knowledge of the world. Another interesting property is the ability of the planner to act so as to remain in the better known parts of the state space when appropriate, *even when its knowledge of the world suggests the existence of better solutions elsewhere*, in order to save on the required computational effort.

More subtle and complex computational control strategies such as these exact a certain cost in overhead, but the increased responsiveness they provide to internal and external environmental factors make them of high value in the development of planning systems potentially scalable to large domains for which classical methods are intractable. This paper demonstrates how careful decision-theoretic reasoning and use of a meta-level control methodology can provide strategies with many of the characteristics of natural problem-solving activity.

The Base-Level Planner

A well-known algorithm exists for finding an optimal policy for Markov decision processes, the *policy iteration* algorithm of Howard (1960). This algorithm starts with an arbitrary policy and incrementally improves the action choices assigned to each state until it reaches an optimal policy. This algorithm would provide an optimal planner if its computational costs for large problems were not so high. However, if the agent must pay a cost for the time it spends computing its plan, running policy iteration on the entire problem space might not be a good strategy, as each iteration of the algorithm requires solving a set of linear equations whose size is on the order of the plan space.

We will therefore consider a planner doing more limited computation before each move. We associate with each state a value, initialized by heuristic information, estimating the distance, and hence the expected cumulative reward, to the goal. (In this paper, we will use value and cost interchangeably, modulo a sign change.) For path-planning and the 8-puzzle, there is a natural underestimate of this distance, the Manhattan distance to the goal, which can be initially assigned to each state. (The Manhattan distance assigns to each position of the path-planning problem its distance from the goal if there were no obstacles or motor errors, and similarly assigns each state of the eight puzzle the sum of the distances each piece would travel to its goal position if it could move without error or obstruction by other pieces.) These value estimates are improved with each computation, and can be maintained across problem-solving episodes (differing possibly in initial state but not in reward function) for cumulative performance improvements. In each such episode, the planner alternately computes to improve the value estimates, and chooses moves using these estimates. (This model differs from those considered by Dean *et al.* (1993), where computation is done either prior to or concurrently with action, rather than interspersed with action. Prior and concurrent deliberation models do not provide the agent with sufficient flexibility in deliberation scheduling to allow for formulation of dynamic tradeoffs between computation time and action quality to the same extent.)

Policy Iteration

Since our algorithm will be based on policy iteration, let us describe it for the restricted case we are interested in, goal-directed planning problems. Following Koenig (1992), we define, given a particular policy, an ergodic set to be a minimal set of states which, once entered, are never left. In general, goal states will be the ergodic sets. Any policy which has an ergodic set which does not consist of a single state could lead the agent to performing an infinite loop, which we will take to be against the agent's interests (for the simple goal-directed utility function described above, such a plan would have maximal negative utility). We will therefore only consider plans for which the agent is always eventually absorbed by some ergodic state. If we start with such a plan, and use a goal-directed utility function, then policy iteration will never generate a plan violating this since it always improves on the current plan.

For such plans, we can define the utility as the expected cumulative reward of the agent (without time discounting). Let us introduce for each state x a variable V_x representing the expected future value for an agent starting in that state and using that plan. These value variables are related by the fact that the value of state x is equal to the expected value of the states it will reach in the next step, plus the reward of being in state x (-1 for the goal-directed utility function):

$$V_x = \sum_y P\left(y|a_x, x\right) V_y - 1 \tag{1}$$

where $P\left(y|a_x, x\right)$ is the probability of reaching state y if action a_x is taken in state x. Finally, absorbing states are assigned their intrinsic values. This gives a set of simultaneous equations that can be solved by standard methods. Using the goal-directed utility function above, these values represent the expected number of state transitions between the agent and its goal using the current plan.

Policy iteration involves computing the state values for the current plan, and then choosing a new action a for each state which maximizes the expected value of the state reached by the action.

$$a = \arg\max_{a'} \sum_y P\left(y|a' x\right) V_y \tag{2}$$

If no action improves the expected value, the old choice is maintained (to prevent oscillation between equal-value policies). This is iterated until no new actions are assigned. Each iteration returns a plan of strictly greater value, and the algorithm is guaranteed to converge on the optimal policy. (For the general case involving larger ergodic sets, there is a similar, but slightly more complicated algorithm).

Local Computation

For an agent under time pressure, policy iteration on the entire state space is too expensive. Our algorithms therefore consider only a neighborhood (the *envelope*) of the current

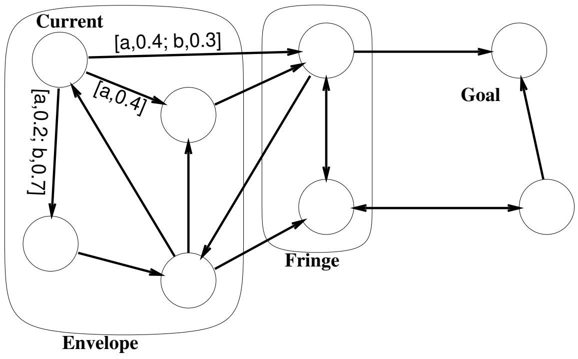

Figure 1: **Markov decision process with local envelope. Actions a, b cause transitions with the probabilities indicated on the arcs.**

state in state space, fixing the values of the surrounding states (the envelope's *fringe*) at their current estimates, and doing policy iteration on the envelope given those values (Fig. 1). The planner starts with an initial policy and heuristic values for the states, and alternates among determining an appropriate envelope, updating its policy and value estimates on this envelope using the Local Policy Iteration Algorithm, and acting according to its current policy. This strategy is similar to (but was developed independently of) those described in the papers of (Dean *et al.* 1993) and (Thiébaux *et al.* 1994); however, its use of estimated values on non-envelope states provides important additional information for determining envelope action choices, allowing for meaningful action choices to be made before a path to the goal is found, and summarizing previous computational effort outside the current envelope. The resulting expected costs of the envelope states are then their expected distance to the fringe plus the fringe's estimated distance to the goal. The cost of time is incorporated into the methods for determining an envelope and choosing when to commit to an action, discussed below.

Local Policy Iteration Algorithm

1. **Fix** values of fringe states at their current estimates, and **assign** stop states their reward.
2. **Assign** an action a to each remaining envelope state x.
3. **Solve** the set of simultaneous equations (1) for the values V_x of these states.
4. **Choose** new best actions a for these states using Eq. (2).
5. **If** any action assignment has changed, **go to** 3.
6. **Return** new value estimates and actions for these states.

Because we store value estimates permanently with each state, we can prove that:

- The value stored in each state approaches the true minimum goal distance and the corresponding plan approaches the optimum as the planner faces (possibly multiple) problem-solving episodes.

To be more specific, let us consider a planner that starts with a heuristic which does not overestimate the expected cost of any state (and gives each non-goal state a cost of at least 1, such as the Manhattan distance), and uses the goal-directed utility function. Before each action, it does at least some computation, and runs policy iteration on its chosen envelope to completion. Then, because policy iteration gives each envelope state a cost of the true expected distance to the fringe plus the estimated fringe distance to goal, by induction no state cost estimate can ever exceed the true distance. For a heuristic such as the Manhattan distance, where neighbors in state space have heuristics differing by 1, computation can only increase the estimated cost of a state. In the worst case of an uninformative heuristic of all 1's, this is also the case. Any further information can only improve performance. No values other than the correct ones are stable; there will in all other cases be some state having a lowest action cost other than 1 plus the expected cost of its neighbors and when it is visited it will be changed. So, in the worst case, values can only go up, and will continue to do so until the correct estimates are reached.

Control of Computation

Because the agent cannot completely predict the results of its actions, but can determine them once action has been taken, it gains information by acting. The situation differs therefore from that faced in deterministic problem-solving, where the computational effort demanded of the agent is independent of when it chooses to execute its actions. Because knowing the results of an action reduces the possibilities necessary to consider later, acting early can save computational effort, and this has to be weighed against the potential improvement in action quality to be gained by delaying for further deliberation. This leads to consideration of the potential utility gain afforded by further computation, which is measured by the amount one expects one's final action choice to be better than one's current choice (as elucidated in (Russell & Wefald 1991)).

Our computational strategy provides a hill-climbing algorithm, maintaining only one plan at a time and considering local changes for its possible improvement. (Standard tree representations for problem solving are too cumbersome in stochastic domains where a state can be visited multiple times by the same plan.) Therefore, when trying to measure the potential gain of further computation, it is only feasible to compare the current plan to possibilities generated by a few local changes to the current plan. Because the action choice in the current state is of the greatest immediate importance, we will consider plans differing from the current one only by this action choice for the purpose of deciding whether and how to continue computation.

Intermediate results in our version of the policy iteration algorithm readily provide the probability $P\left(f|y\right)$ of reaching any given fringe state f first from any given envelope state y by following the current plan. Therefore, when provided with an estimate σ_f of how much a fringe state's estimated value is likely to vary as a result of further computation, we can easily determine the resulting value variance $\sigma_{y|f}$ for any envelope state y if the current plan is kept.

$$\sigma_{y|f} = P\left(f|y\right)\sigma_f \tag{3}$$

If variance estimates are not available (or all assumed to be the same), the fringe state most likely to be reached first from a given envelope state has the greatest potential impact on that state's value. Both (Dean *et al.* 1993) and (Thiébaux *et al.* 1994) therefore choose the fringe state most likely to be reached first from the current state as the best candidate for further envelope expansion. However, the values of the current state's neighbors y have the greatest impact on the action to be chosen in the current state x, so the fringe state f whose variance is most likely to affect the current action choice (through the neighbors' values) is the best one to expand. The potential gain G_{fx} in expanding it is how much better than the current action the ultimately chosen action is expected to be.

$$G_{fx} = \max_{a'}\left[\sum_y P\left(y|a'x\right)\left(V_y + \sigma_{y|f}\right) - 1\right] - \left[V_x + \sigma_{x|f}\right] \tag{4}$$

(This is actually an underestimate of the value of expanding a fringe state, because it ignores indirect improvements to the action choice resulting from changes in other states' action choices; accounting for these effects is hindered by our localized plan representation. The definition of G_{fx} could also be generalized by integrating over a distribution for σ_f instead of using a point estimate.) We thus have an analytic strategy for choosing envelope expansions (unlike the trial and error learning approach of (Dean *et al.* 1993)), and expand the envelope according to impact on action choice quality.

The amount of variance expected of a fringe state is of course a function of the amount of computation done on a neighborhood of it, and the time cost of the computation similarly grows with amount. The agent should do computation in the amount determined by optimizing the difference between expected gain and time cost. If the agent is only given the choice between doing a fixed size computation or not (with determined values for time cost and expected value variance), the decision hinges on a comparison of expected action quality gain to a fixed cut-off.

Let us summarize the computational control structure. After an action, the agent determines its current state (which is its initial envelope) and grows the envelope by moving fringe states into it whose expected contribution to action choice quality is highest. At occasional intervals (e.g. after every five additions) policy iteration is run on the envelope. This continues until the best fringe state fails to offer an expected gain greater than the cost of policy iterating on the expanded envelope, at which point policy iteration is run a last time, and the agent takes the action from its current state which leads to the highest expected resulting state value.

Even using a simple, uniform variance estimate (all σ_f's the same), the algorithm has several interesting properties. Because envelope expansion is determined by a direct comparison between expected gain and computational time cost (Line 7), any increase in the cost associated with time spent on computation, relative to the unit cost of physical action, increases the potential value gain a computational plan improvement must achieve in order to be done. Therefore,

The Base-Level Planning Algorithm

1. **Initialize** value V_x of all states x using the heuristic (can be done as value becomes needed), and **initialize** current-state.
2. envelope $\leftarrow$ current-state.
3. fringe $\leftarrow$ neighbors of current-state.
4. more-comp $\leftarrow$ yes.
5. **For** $i = 1$ **to** number-of-envelope-expansions-per-policy-iteration (e.g. something like 5)
6.   **do** best-fringe $f \leftarrow \arg\max_{f' \in fringe} G_{f',current\text{-}state}$ (determining G by Eq. (4))
7.   **If** $G_{f,current\text{-}state} >$ time-cost(size(envelope))
8.   **do** envelope $\leftarrow$ envelope $\cup$ best-fringe
9.   fringe $\leftarrow$ neighbors of envelope
10.   **else do** more-comp $\leftarrow$ no.
11. **Run Local Policy Iteration Algorithm** on envelope to **update** V_x for non-ergodic envelope states x.
12. **If** more-comp = yes **then go to** 5.
13. **Do action** a as recommended by Eq. (2) for current-state: current-state $\leftarrow$ result of action.
14. **Unless** current-state is a stop state, **go to** 2.

- The planner automatically adjusts the time spent in look-ahead vs. physical exploration in response to time pressure.

  Boundary case: As time cost $\rightarrow 0$, the algorithm increases envelope size if any action could be better than the present one.

  Boundary case: As time cost $\rightarrow \infty$, the algorithm updates only the current-state value between moves, the minimum necessary to guarantee convergence to optimal plans.

This behavior follows analytically from the algorithm definition. We have shown empirically that an agent will complete a run with fewer computational time steps, at the expense of a longer path, when the cost of time is high. Fig. 2 shows the tradeoffs obtained when running the algorithm on navigation domains like that shown in Fig. 4. The points are labeled with the cost assigned to one time step, in move counts. Policy iteration on an envelope of size n was assessed n^3 time steps. The times and moves were accumulated over a set of 72 runs over various domains using a probability of random action failure of 0.1, and a σ_f of -1.0.

Also, in a domain with greater randomness in action results, the ability to predict which fringe state an agent will hit first (and hence the highest fringe probability) is lowered, reducing the expected effect of further computation on action choice quality.

- The agent will respond to greater randomness by expending less effort on prediction.

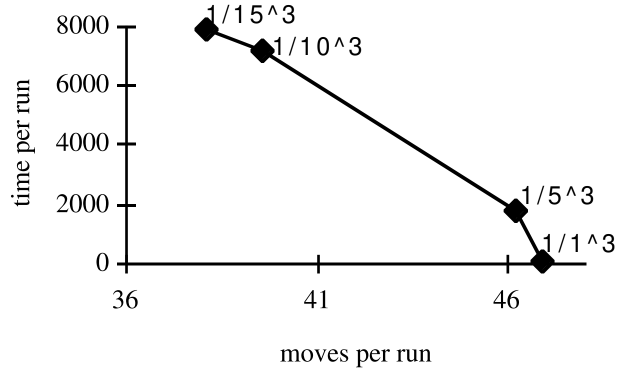

Figure 2: **Responsiveness of the Base-Level Planning Algorithm to the cost of time**

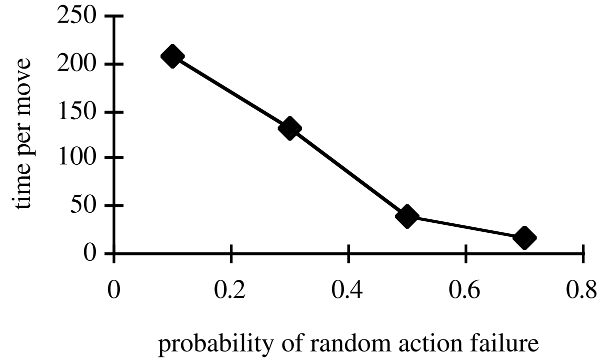

Figure 3: **Responsiveness of the Base-Level Planning Algorithm to randomness in the environment**

Boundary case: As transitions become totally random, the algorithm updates only the current-state value.

This follows because as the $P\left(y|a'x\right)$ become uniform in a', the $\max_{a'}$ in Eq. (4) cannot differ much from the value for a, forcing G_{fx} towards 0. This behavior is demonstrated for the navigational domain in Fig. 3 (run as above, using the lowest time cost).

The Introspective Planner

A planner implementing the system described so far will exhibit appropriate responsiveness to several aspects of its environment in controlling its computational effort. It will move where its model of the world indicates a shortest path, and it will devote its computations to where they will have greatest impact on its action choices according to its world model. However, this system does not incorporate control strategies making use of the agent's knowledge of what it knows. Such strategies would allow the agent to, for example, expend computational effort where its knowledge most needs improving, and move along well-known (albeit perhaps suboptimal) paths in order to avoid extensive computation. A planning agent able to follow such strategies can be called *introspective*, because its decisions are based not only on its knowledge of the world, but on its knowledge about its knowledge of the world.

Introspective planning techniques are an extension of common metalevel control techniques (discussed in, for example, (Russell & Wefald 1991)). Computations such as those performed by the base-level system described above are controlled somewhat like physical actions by a higher level controller. This architecture can be iterated, getting higher level controllers allowing for deeper forms of introspection. This section will limit discussion to a minimal extension of the base-level system, which will be enough to express the introspective strategies mentioned above.

Representation of Computational Knowledge

In order to use introspective knowledge, the planner needs some representation of what base-level computational work has been performed. We will extend the representation of each state to include a parameter indicating how much computational effort has been expended in updating the estimated value of that state. How best to characterize this effort is not obvious, but a parameter such as the sum of the time costs of each policy-iterated envelope which included the state will certainly provide useful information. We will call such a parameter a *knowledge heuristic* k to distinguish it from the *physical heuristic* p estimating minimal distance to the goal. While more detailed representations are obviously desirable, knowledge heuristics turn out to be quite effective.

One use for the knowledge heuristic is for conditioning the estimate of expected variance of a state's physical heuristic with further computation. The amount of computation previously done on a state is highly relevant to the expected consequences of further computation. For example, repeating computations done previously will make no change to the physical heuristic. One expects that as k grows for a given fringe state f, σ_f will decrease. It follows from Eqs. (1), (3) and (4) that $G_{fx} \leq \sigma_f$, so

- The planner's choice of envelope expansion is sensitive to its previous computational efforts in that direction.

Boundary case: As $k \to \infty$, $\sigma_f \to 0$, so $G_{fx} \to 0$ and the algorithm does not expand f.

A particular variance estimate, represented as a function of the knowledge heuristic, is discussed in the Experimental Results section below. Using these variance estimates as described above for deciding where and whether to expend further computational effort will enable the system's computational efforts to focus more on where its knowledge is in greatest need of improvement.

Total State Value

The introspective planner works by associating with states a value which reflects both estimated action costs *and computational costs* to reach the goal. This value will fill many of the roles filled by the physical heuristic for the base-level system, allowing for the tradeoffs between these costs desired of an introspective agent. Which roles this total value can fill are constrained by coherence requirements which we will analyze below.

The total value $T(k,p)$ associated with each state will be a function of the two heuristics. It should estimate the sum of the remaining distance and computational costs to the goal. An example of such a function is discussed in the Experimental Results section. Note that if the total cost takes on the value of the physical heuristic, the system's decisions will remain independent of its knowledge about previous computational effort, and the base-level system behavior will be reproduced.

Knowledge of a total cost function is of use to the agent in places where it captures the criteria relevant to control decisions better than the physical heuristic alone. For example, when the agent decides to act, it should choose the action a leading to the lowest expected total cost among its neighbors y, rather than minimizing the expected resulting physical heuristic.

$$a = \arg\max_{a'} \sum_y P\left(y|a'x\right) T\left(k_y, p_y\right) \tag{5}$$

Such an action minimizes the remaining total cost to goal. Because the total cost reflects computational costs,

- The agent's action choices reflect the value of its knowledge of the world.

  Boundary case: As time pressure increases, $T(k,p)$ becomes dominated by computation costs, so the agent will act to minimize future computation at the expense of distance.

This behavior is exhibited experimentally below.

Another point is that whereas the physical heuristic is an (under)estimate of the true ("God's-eye-view") minimal distance to goal, the total cost function can model the physical distance that will be realized by the agent's *actual* control strategy, so

- The agent's action choices are based on the realizable (not theoretically minimal) distance to goal.

This captures an important distinction, one used in (Baum 1992) to improve game playing performance. (Of course, over repeated trials, these distances will converge.)

This change in action choice criteria should be reflected in the criteria used for choosing computations as well. The impact of expected physical heuristic variance in a fringe state can be propagated to the current state's neighbors as before, where its effect on the neighbors' total value can be determined by finding the total cost associated with the varied physical heuristic and the knowledge heuristic which would result from doing the computation. These values are then used to evaluate the worth of the current-state actions, so that the expected gain in action quality can be computed.

$$\begin{aligned} G_{fx} &= \max_{a'} \left[\sum_y P\left(y|a'x\right) T\left(k_{y,\,{}^{post}_{comp}}, p_y + \sigma_{y|f}\right) \right] \\ &\quad - \left[\sum_y P\left(y|ax\right) T\left(k_y, p_y + \sigma_{y|f}\right) \right] \end{aligned} \tag{6}$$

The Introspective Planning Algorithm

Modify The Base-Level Planning Algorithm as follows:

(a) **Replace** Eq. (2) with Eq. (5) **in Line** 13.
(b) **Replace** Eq. (4) with Eq. (6) **in Line** 6.
(c) **Replace** V_x with p_x **in Lines** 1 **and** 11.
(d) **Add** 1a. **Initialize** $\sigma_f(k,p)$ and $T(k,p)$.
(e) **Add** 11a. **Update** k for all states in envelope.
(f) **Add** learning steps to **Update** $\sigma_f(k,p)$ and $T(k,p)$.

Note that the primary computational effort, the one estimated by the knowledge heuristic and hence used for conditioning the total value function, is still that devoted towards updating the physical heuristic using policy iteration on local envelopes. One might assume that the total value should replace the physical heuristic in these computations, but such an approach would lead to several problems. For one, the relation between the total values of neighboring states is not as simple as that for physical distances. The immediate physical reward of moving to a state is just -1, but the one-move total reward depends on the amount of computation done as well. Policy iteration minimizes cumulative costs using fixed transition costs; it would be inconsistent to apply it when the transition costs vary as a function of using policy iteration itself. Such concerns force a system to differentiate between the computational efforts it is trying to control and those used in this control process.

Experimental Results

The Introspective Planning Algorithm has been implemented and tested on the randomized 8-puzzle with a probability of random action failure of 0.2 and a time step cost of $1/10^3$. Remaining time cost to goal and fringe variance were assumed to drop inversely with the knowledge heuristic. All runs were conducted with random initial states of distance about 10 from the goal. An initial set of 80 runs was conducted to learn good heuristics near the goal and to estimate the numerical coefficients (using the temporal difference learning methods of (Sutton 1988)) in the equations for variance and total value:

$$\sigma_f = -\frac{1}{2(10k+1)} \tag{7}$$

$$T(k,p) = -p - \frac{p/3}{\frac{3k}{p}+1} \tag{8}$$

(The time cost term of $T(k,p)$ was assumed to grow linearly with p at $k=0$, and $\partial T/\partial k|_{k=0} = 1$ so computational time expenditures reduce future costs as expected.)

Next, 40 runs each of the Base-Level Planning Algorithm (using $\sigma_f = -1/4$, as a typical value for k was 0.1) and the Introspective Planning Algorithm were conducted. Because of the possibility of random action failure, the agent occasionally moves away from the goal (and hence away from the well-explored portion of the state space). Such actions often take the agent to states where following the minimal physical heuristic leads in a direction away from the goal, towards an artificial minimum in the Manhattan distance. An agent following such a path will have a long solution route to the goal (more than 40 moves, instead of $\sim$13). This occurred for the Base-Level Planning Algorithm in 8 of its 40 runs. Because introspection provides a force towards staying in the explored portion of the state space, the Introspective Planning Algorithm was only tempted by a false minimum in 1 of its 40 runs.

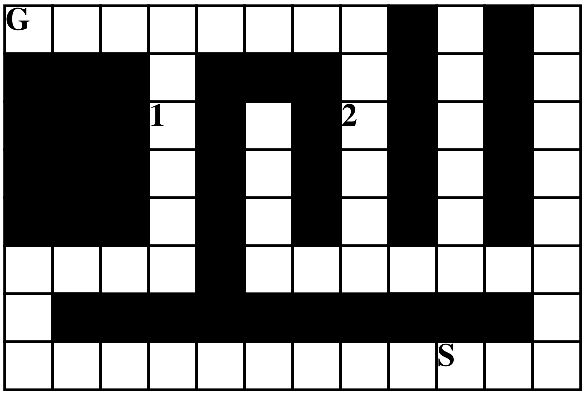

Figure 4: **Navigation Domain.**
Agent is to move from start (S) to goal (G).
Note that route 1 is longer than route 2 but a move towards 1 from S reduces the Manhattan distance heuristic more.

The Introspective algorithm has also been tested on the navigational domain shown in Fig. 4, with a random action failure probability of 0.05. On its first run, the agent takes route 1 to the goal, as moving left from the start state most reduces its estimated physical distance. As it continues to take route 1, it improves this estimate towards the true physical distance, which is larger than its estimate of the physical distance to the right of the start state. The Base-Level algorithm therefore quickly begins moving right, exploring the upper right region and discovering the shorter route 2. If the cost of computation time is high, however, the Introspective algorithm will continue to take the known route 1 so as to avoid the computational burden involved in exploring the unknown upper right region, as shown in Table 1.

Time Cost:	$1/15^3$	$1/2^3$
Base-Level	2.7	2.3
Introspective	2.0	7.0

Table 1: **Average number of runs before taking route 2**

Conclusions

This paper has extended classical methods for determining good policies for Markov decision processes by adding explicit consideration of the cost of the time spent in computing policies. The method bears a general resemblance to other localized extensions of dynamic programming, such as Incremental Dynamic Programming (Sutton 1991) and the work of (Dean *et al.* 1993) and (Thiébaux *et al.* 1994), but adds a new analysis of the factors appropriate for envelope determination: computational effort is put where it will make the most difference to quality of action choice, and is expended only so long as it will make a significant difference to the next action choice. This analysis is then extended to take advantage of information the agent can accumulate about its previous expenditures of computational effort. The resulting control strategy exhibits many interesting behaviors, mirroring considerations that would be of importance to human problem-solvers.

The planner makes good use of heuristic information about the problem space. If the heuristic underestimates costs, it provides a force for (simulated) state space exploration when the computational cost is not too expensive. Under such conditions, one can guarantee convergence to an optimal plan over repeated trials. The methodology also provides for concise storage (in the form of two heuristic values for each state) of the results of previous planning efforts for use in future encounters with the same state space region, and the planner has the anytime property of (Dean & Boddy 1988) in that its policy improves as a function of the time it has had to compute.

The overhead incurred by the proposed control mechanisms grows more slowly with computation size than the cost of policy iteration, so the computational savings for large spaces are not swamped by control costs.

The planner's computation and action choices are responsive to time pressure and randomness, as well as to the status of the agent's current knowledge and the possibilities and costs of changing it. The resulting flexibility of the planner's control structure enables better focusing of computational effort, providing features needed in planners for use in environments too large for the more extensive computations required by conventional methods.

References

Baum, E. B. 1992. On optimal game tree propagation for imperfect players. In *AAAI-92*, 507–512.

Dean, T., and Boddy, M. 1988. An analysis of time-dependent planning. In *AAAI-88*, 49–54.

Dean, T.; Kaelbling, L. P.; Kirman, J.; and Nicholson, A. 1993. Deliberation scheduling for time-critical sequential decision making. In *UAI-93*, 309–316.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Cambridge, Mass: MIT Press.

Koenig, S. 1992. Optimal probabilistic and decision-theoretic planning using Markovian decision theory. Technical Report UCB/CSD 92/685, UC Berkeley, Calif.

Russell, S., and Wefald, E. 1991. *Do the Right Thing*. Cambridge, Mass: MIT Press.

Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. *Machine Learning* 3:9–43.

Sutton, R. S. 1991. Planning by incremental dynamic programming. In *8th Intl Wkshp on Machine Learning*.

Thiébaux, S.; Hertzberg, J.; Shoaff, W.; and Schneider, M. 1994. A stochastic model of actions and plans for anytime planning under uncertainty. *Int. J. of Intelligent Systems*.