

Dynamic non-uniform abstractions for approximate planning in large structured stochastic domains

J. Baum and A.E. Nicholson
School of Computer Science and Software Engineering
Monash University, Clayton, VICTORIA 3168, Australia
phone: +61 3 9905-5225 fax: +61 3 9905-5146
`{jirib,annn}@cs.monash.edu.au`

Abstract

The theory of Markov Decision Processes (MDPs) provides algorithms for generating an optimal policy. For large domains these algorithms become intractable and approximate solutions become necessary. In this paper we extend previous work on approximate planning in large stochastic domains by using automatically-generated non-uniform abstractions which exploit the structure of the state space. We consider a state space expressed as a cross product of sets, or dimensions. We obtain approximate solutions by varying the level of abstraction, selectively ignoring some of the dimensions in some parts of the state space. We describe a modification of a standard policy generation algorithm for the now non-Markovian decision process, which re-calculates values for nearby states based on a locally uniform abstraction for each state. We present methods to automatically generate an initial abstraction based on the domain structure and to automatically modify the non-uniform abstraction. The changes to the abstraction are based on both the current policy and the likelihood of encountering particular states in the future, thereby taking into account the agent's changing circumstances.

Keywords: Markov Decision Processes, approximate algorithms, non-uniform abstraction, dynamic algorithms, planning under uncertainty

1 Introduction

In a deterministic world where an agent can be certain of the consequences of its actions, it can plan a sequence of actions, knowing that their execution will necessarily achieve the goal. While such an assumption may be applied to industrial robots in very limited domains, it is not appropriate for flexible, multi-purpose robots and other intelligent software agents. These agents must be able to plan in the dynamic, stochastic domains in which they will operate.

For small to medium sized stochastic domains, the theory of Markov decision processes provides algorithms for generating the optimal plan [1, 2]. However, as the domain becomes larger, these algorithms become intractable and approximate solutions become necessary [9, 7]. In particular where the state space is expressed in terms of dimensions, or as a cross product of sets, its size and the resulting computational cost is exponential in the number of dimensions. On the other hand, fortunately, this results in a fairly structured state-space where effective approximations often ought to be possible. Dearden and Boutilier use this structure [5, 3] to obtain an exact solution or [4, 8] an

approximate solution by ignoring particular dimensions, however, their abstractions are fixed throughout execution; literals are also deleted from the problem in a pre-determined sequence. The solution presented in this paper is based on selectively ignoring some of the dimensions, in some parts of the state space, some of the time. In other words, we obtain approximate solutions by dynamically varying the level of abstraction in different parts of the state space.

The agent architecture consists of a separate planner and executive; the planner continually improves and modifies the approximation while the agent executes actions. Dean *et al.* [7] describe a locality-based approximation for such an architecture, where the agent can expend more of its planning on the most likely paths and on the closer future, expecting that when and if it reaches other parts of the state space, it can improve the approximation as appropriate. We use the same idea with our abstraction-based approximation.

This paper deals with control error exclusively. Sensor error is not considered and it is assumed that the agent can accurately discern the current world state. It is also assumed that the agent accurately knows the state space, the goal or reward function, and a distribution over the effect of its actions.

The remainder of this paper is organised as follows. In Section 2 we briefly explain classical planning in MDPs and the structure of the agent, and present an example domain in Section 3 to facilitate explanation. Section 4 presents the idea of non-uniform abstraction, and discusses how it might be relatively simply represented. Section 5 deals with the modifications to classical MDP planning necessitated by this approximation, involving re-calculation of values based on a locally uniform abstraction. In Section 6 we present methods to automatically generate an initial abstraction based on the domain structure and to automatically modify the non-uniform abstraction. The changes to the abstraction are based on both the current policy and the likelihood of encountering particular states in the future, thereby taking into account the agent's changing circumstances. Finally, we discuss the approach and indicate directions for future research in Section 7.

2 Planning in Stochastic Domains

One approach to planning in stochastic domains involves using Markov Decision Processes (MDPs). A MDP is a tuple $\langle S, A, T, R, s_0 \rangle$, where S is the *state space*, A is the *set of available actions*, T is the *transition function*, R is the *reward function* and $s_0 \in S$ is the *initial state*. The agent begins in state s_0 . At each time step, the agent selects an action $a \in A$, which, together with the current state, indexes T to obtain a distribution over S . The current state at the next time-step is random according to this distribution. The agent is also given a reward at each time step, calculated by R from the current state (and possibly also the action selected). The aim of the agent is to maximise some cumulative function of these rewards, typically the expected discounted sum. In a fully-observable MDP, the agent has full knowledge. In particular, the agent is aware of T , R and the current state when selecting an action. It is well-known that in a fully-observable MDP, the optimal solution can be expressed as a policy

$\pi : S \rightarrow A$ mapping from the current state to the optimum action. The planning problem, then, is the calculation of this π .

The standard algorithms for computing the policy are policy iteration [11] and value iteration [1]. Each consists of two iterative interrelated parts, one dealing with changing the policy, the other with computing the values. The iterative steps are:

$$\pi(s) \leftarrow \arg \max_a \sum_{s'} \Pr(s, a, s') V(s')$$

and

$$V(s) \leftarrow R(s) + \gamma \sum_{s'} \Pr(s, \pi(s'), s') V(s')$$

where γ the discounting factor is a parameter ($0 \leq \gamma < 1$), the probabilities are specified by T , and $V : S \rightarrow \Re$ is the *value function*, calculated as a byproduct of the algorithm. The value function gives the expected discounted sum of reward for each state.

In policy iteration each step is iterated in turn repeatedly; there is an alternating sequence of ‘policy calculation’ and ‘value calculation’, which use the first and second update formula respectively, and each separately iterates until convergence. In value iteration they are iterated together; for each state both are applied in order. Depending on the algorithm variant, the iteration may be applied to all states simultaneously (using a copy of V) or in an arbitrary or specific sequence.¹ In this paper we use the value iteration algorithm.

As S becomes larger, the calculation of the optimal policy, π_{opt} , becomes more computationally expensive. Since in practice the amount of computation allowed to the agent is limited (either strictly, or by incurring a penalty), this necessitates some approximations in the process. One type of such approximations is to approximate the MDP with an MDP with a smaller state space E . For example, Dean *et al.* [7] use a subset of S , augmented with a special state OUT representing the remainder of S . This is based on notion of locality, and plans only for those states that are likely to be reached. $\pi(\text{OUT})$ is specified by the designer.

If the state space has some structure, however, the planner ought to be able to do better by taking advantage of this structure to focus the computation and improve the approximations. Often, the state space can be represented as a cartesian product of dimensions, each of which is a set containing the values that dimension may take. This makes $|S|$ exponential in the number of dimensions. For example, even in a relatively simple navigation environment, there may be a dozen doors, each of which may be at least open or closed, increasing the size of the domain several-thousand-fold compared to an ‘open-plan’ layout without doors. Planning in S quickly becomes intractable.

One possibility is to make E the cartesian product of a subset of these dimensions [8, 15]. Excluding a dimension from the envelope E in this way is called ‘uniform abstraction’ or simply ‘abstraction’, and the more dimensions are excluded, the more abstract E is. It is also possible to exclude different

¹See [17] for an example of such a specific sequence.

dimensions from different parts of E — this is called ‘non-uniform abstraction’, and is the focus of the research presented in this paper.

2.1 Agent Structure

We assume an agent architecture which Dean *et al.* call *recurrent deliberation*, where the agent is split into a *planner* and an *executive*, and the agent has processing power available while it is acting. This means the agent does not need to plan so well for unlikely possibilities, and can therefore expend more of its planning on the most likely paths and on the closer future, expecting that when and if it reaches other parts of the state space, it can improve the approximation as appropriate. The planner attempts to continually improve the approximate policy, either by undertaking additional computation in the iterative policy-generation algorithm, or by updating the focus based on the latest observation of the current state. For example, the agent may have originally planned with s_0 in mind, concentrating on producing a good policy for those parts of S it is likely to visit. If, due to misfortune, it finds itself in some part of S which it did not expect, it can attempt to improve that part of the policy. Conversely, once it has moved some way from s_0 and does not expect to visit it again, it can discard the part of the policy surrounding s_0 to speed up policy computation. A similar architecture is used by the CIRCA system [14, 10] to guarantee hard deadlines; in CIRCA terminology, the planner is the AIS, and the executive is the RTS.

The algorithm for the planner is shown in Figure 1. The key steps are Step 1 and Step 3d, for the initial choice and modification of E almost exclusively determine the quality of the approximation. In this paper we look at how these steps may be done using non-uniform abstraction; the other parts of the algorithm require only minor changes.

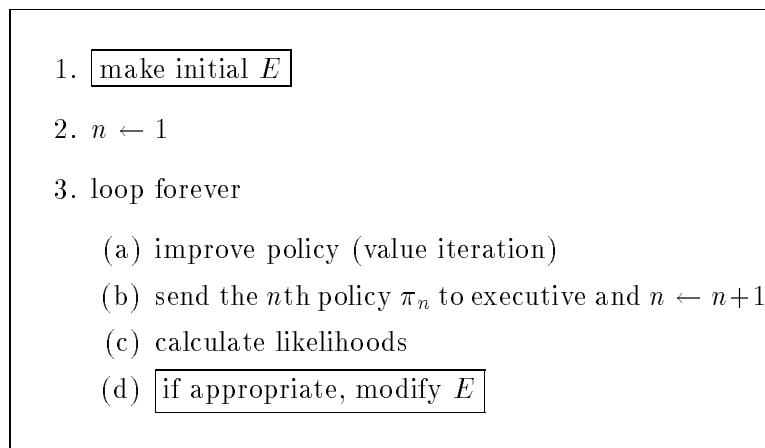


Figure 1: Planner algorithm

3 Example domain

The example domain used in this paper is a simple robot navigation world, similar to the RN1 world of [12] but with the addition of two walls and three doors (see Figure 2(a)).

It consists of a 10×10 grid, with one wall between $y = 2$ and $y = 3$ and the other, for $y \geq 3$, between $x = 4$ and $x = 5$. The doors are (in order) south of $\langle 2, 2 \rangle$ and $\langle 7, 2 \rangle$ and east of $\langle 9, 4 \rangle$; each can be either **open** or **closed**. There is also a ‘damaged’ dimension dmg , which may take the values **true (T)** or **false (F)**, which indicate whether the robot or some other part of the world (a door, or wall) has been damaged; this is used to discourage the agent from attempting to walk through a closed door.² More formally,

$$S = S_x \times S_y \times S_{d1} \times S_{d2} \times S_{d3} \times S_{dmg}$$

where $S_x = S_y = \{0, 1, \dots, 9\}$, $S_{d1} = S_{d2} = S_{d3} = \{\text{open}, \text{closed}\}$ and $S_{dmg} = \{\mathbf{T}, \mathbf{F}\}$. Note that the full state space of this example contains $|S| = 10^2 \times 2^3 \times 2 = 1600$ states. The initial state consists of the agent in the location marked s_0 with no damage and all doors closed. More formally, $x = y = 0$, $d1 = d2 = d3 = \text{closed}$ and $dmg = \mathbf{F}$.

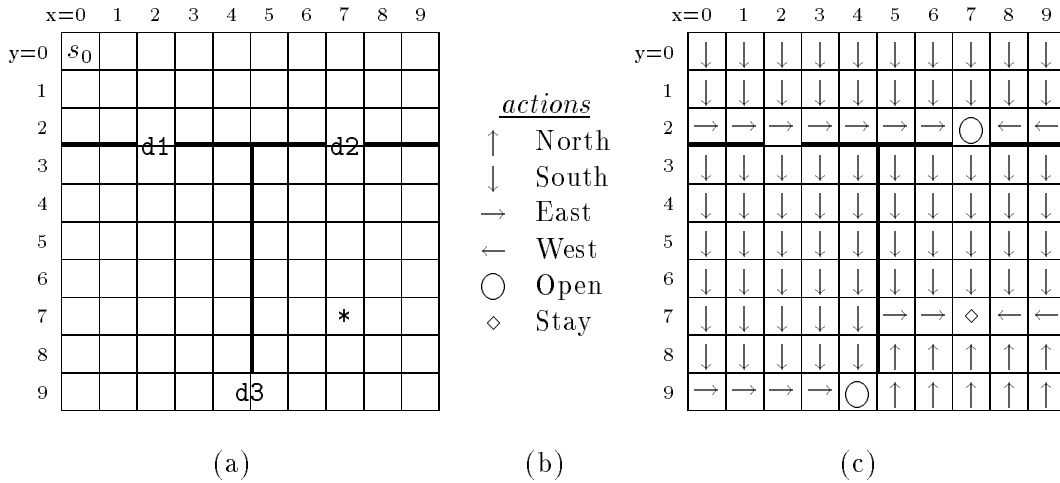


Figure 2: Example domain. (a) Initial situation with starting location s_0 , goal $*$, all three doors ($d1$, $d2$ and $d3$) **closed** and no damage; (b) Graphical representation of actions; (c) Part of an optimal policy obtained using value iteration on the complete state space, with $\gamma = 0.99999$.

There are six actions available in A (see Figure 2(b)): movements in all four cardinal directions, **OpenDoor** and **Stay**, which is a do-nothing action. The **OpenDoor** action opens the immediately adjacent door; attempting it elsewhere has no effect. Each action may fail with some specified probability. There is also an *a priori* probability distribution associated with each dimension; for

²This is important due to the lack of performance guarantees — the agent might be following a policy for longer than anticipated and reach parts of the state space that are not well planned for.

the examples used throughout this paper, we use uniform *a priori* distributions (e.g. the probability that a door will be `open` is taken to be 50%).

The reward function R corresponds to a goal of achieving a particular location while avoiding damage. It gives 0 for the goal state, -2 for damage and -1 for all other states. Note that with these particular values for R and the *a priori* probabilities, the agent will not attempt to walk through a door unless it is certain that it is open, even if that means failing to achieve the goal. On the other hand, it might, say, move east without checking if it will hit the eastern wall. The reward necessary to force the agent to definitely avoid damage (corresponding to a CIRCA [14] control-level goal) depends on the *a priori* and transition probabilities. However, since the intended domains of application do not have guaranteed actions, guaranteed control-level goal satisfaction will usually not be possible anyway.

For the examples described in this paper, the discounting factor γ has value 0.99999. An optimal policy for the grid locations with all doors closed, no damage and the goal at $\langle x = 7, y = 7 \rangle$ is shown in Figure2(c).

4 Non-uniform abstraction

Non-uniform abstraction is based on the intuitive idea of ignoring some dimensions in *some parts* of the state space. For example, a door is only of interest when the agent is about to walk through it, and can be ignored in those parts of the state space which are distant from the door. In a member of the envelope E , each dimension is either taken into account completely, or ignored altogether. For example, there might be a state in the envelope corresponding to the agent standing immediately north of door $d2$, with $d2$ `closed`, no damage and any position for the other doors. Here, the location, position of $d2$ and damage are taken into account, and the other doors are ignored. Note that the domain is still fully-observable. This is not a question of lack of knowledge about the dimensions in question, but wilful conditional ignorance of them during planning as a matter of computational expediency.

We will use the following terminology in the remainder of this paper. Members of E are called *envelope states*. Members of S are *specific states*. E is a partition of S , which means that each envelope state is a set of specific states.

4.1 Representation of the envelope

Since E is a partition of S , an envelope state $e \in E$ is a subset of S . Therefore the representation of the envelope states parallels the structure of S . As S is a cross product of several state space dimensions, so each e is represented as a cross product of several envelope state dimension. In the example domain, each e is $e_x \times e_y \times e_{d1} \times e_{d2} \times e_{d3} \times e_{\text{dmg}}$

Each envelope dimension is either (1) a singleton subset of the corresponding state space dimension; the dimension is then taken into account, that is, the state is *concrete* in the dimension; or (2) the entire state space dimension; the dimension is then ignored, that is, the state is *abstract* in that dimension. Note

that when state e is abstract in one or more dimensions, it effectively groups a number of specific states in S .

For example the envelope state having $x = 7$, $y = 2$, $d2 = \text{closed}$, $dmg = \mathbf{F}$ and ignoring the dimensions $d1$ and $d3$, would be represented as $\{7\} \times \{2\} \times S_{d1} \times \{\text{closed}\} \times S_{d3} \times \{\mathbf{F}\}$. This envelope state groups together four specific states, corresponding to the four combinations of `open` and `closed` for $d1$ and $d3$. For brevity, we write this state as $\langle 7, 2, ., \text{closed}, ., \mathbf{F} \rangle$, dispensing with the set notation and using a period to denote an ignored dimension. When an envelope state e is concrete in all dimensions, $|e| = 1$ and e corresponds to exactly one specific state, that is $e = \{s\}$, $s \in S$. In the degenerate case where the envelope consists of a single envelope state e abstract in all dimensions (i.e. ignoring everything), e groups all the specific states together so $e = S$, $E = \{S\}$. It is not inherent in this representation that E is a partition of S ; this needs to be maintained as an invariant through all manipulations.

In some cases, it is necessary to have a precise mathematical measure of the abstractness of a state. For this, we use the *a priori* probability of a state. This is an overall distribution over the states in the state space, without regard to the initial state or the agent’s policy. It makes sense to specify them on a per-dimension basis; the *a priori* probability of a specific state $\text{Pr}(s)$ is then the product of the *a priori* probabilities of its dimensions. The *a priori* probability of an envelope state $\text{Pr}(e)$, on the other hand, is the product of the *a priori* probabilities of its *concrete* dimensions, or, equivalently, the sum of the *a priori* probabilities of its members. If the *a priori* probabilities are uniform, as they are in the example, $\text{Pr}(s)$ is the same for all s and $\text{Pr}(e) = |e| \text{Pr}(s)$.

Note that, when the policy is handed to the executive, the envelope can be compiled into an Finite State Machine (FSM) for fast policy lookup.

4.2 Tree domain representation

As the intended domains of application have a large S , it will not be possible to specify the transition function T directly due to its size. The transition probabilities must therefore be specified in some other, more compact manner. In the current implementation, we specify the parts of T related to the different actions separately. The transition function probabilities for each action are written in the form of a decision tree [16], enriched to allow non-determinism by allowing some nodes to decide at random rather than based on a particular dimension. Allowing non-determinism at multiple internal nodes simplifies the construction of the trees for the abstracted state space.

The reward function R , is specified as an ordinary decision tree, and depends upon the current state only. No non-determinism is required, even for the abstracted state space, since the leaves can simply be combined linearly (averaged).

5 Policy Generation

Given an envelope of non-uniformly abstracted states, we can compute a policy using an algorithm similar to the standard value iteration (or policy iteration)

algorithm described in Section 2. Note that this section pre-supposes the envelope; we discuss how to go about devising this envelope automatically in Section 6. The policy π becomes a mapping $E \rightarrow A$ and the value function becomes $V : E \rightarrow \mathfrak{R}$. The equations then read:

$$\pi(e) \leftarrow \arg \max_a \sum_{e'} \Pr(e, a, e') V(e')$$

and

$$V(e) \leftarrow R(e) + \gamma \sum_{e'} \Pr(e, \pi(e'), e') V(e')$$

where $R(e)$ and $\Pr(e, a, e')$ are specified by the abstracted reward function and abstracted transition function, respectively. Before the policy is used by the executive, it must be translated back into $S \rightarrow A$ form; this is done by simply setting $\pi(s) = \pi(e)$ whenever $s \in e$.

Note that there are likely to be problems as the non-uniformly abstracted approximation will not in general satisfy the Markov property. However, the algorithm can be run, though it will sometimes compute incorrect values and hence return non-optimal policies. The main anomaly, which we call the *ostrich effect*, is discussed below.

5.1 Ostrich effect

When the agent plans to move from a relatively concrete state s_1 to a more abstract state s_2 and thence to s_3 , it may need to know that s_2 was reached from s_1 in order to identify the correct s_3 , which violates the Markov property. If the abstracted approximation is simply treated as an MDP in which the agent doesn't know which s_3 it will reach, it won't correspond to the underlying process, which might reach a particular s_3 deterministically. The problem is especially obvious when the actual s_3 that will be reached is in fact s_1 — the planner will plan a loop. This is reminiscent of a problem noted in [6], where a plan derived from a POMDP failed — the actual robot got into a loop — when, in a particular situation, a sensor was completely reliable.

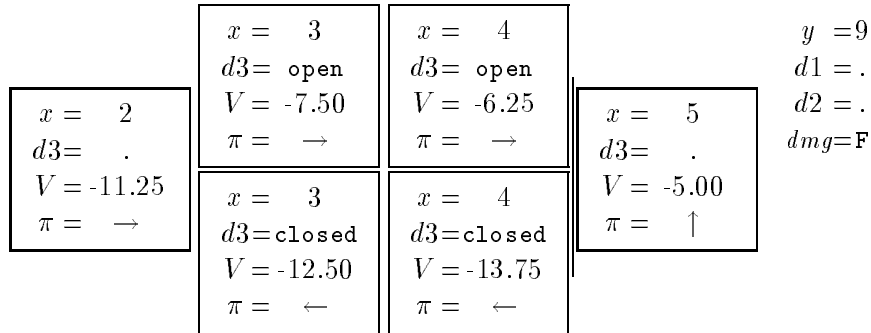


Figure 3: Ostrich effect

Let us consider a specific example of this behaviour. In the example domain, the doors are sufficiently difficult to open that the agent will prefer to move from

a location where a door is definitely closed, to a more abstract region, where the door is open with the *a priori* probability of 50%. Figure 3 shows a small part of a sample E , part of the southernmost row in the vicinity of $d3$. Each box represents one envelope state, showing the x and $d3$ dimensions, the value V and the policy π for that state. For clarity, the other dimensions ($y = 9$, $d1 = .$, $d2 = .$, $dmg = \mathbb{F}$) are omitted from the diagram. Since the goal is to the East of door $d3$, an optimal policy would be the \rightarrow action at all the depicted states except the one with $x = 4$ and $d3 = \text{closed}$, where the optimal action is \circ to open the door. The door, which is between $x = 4$ and $x = 5$ (suggested by a vertical line in the diagram), is relatively difficult to open, with only a 10% probability of success. On the other hand, when moving from $x = 2$ (s_2 in the general description) to $x = 3$ (s_3) the *a priori* probability that the door is already open is 50%. When the calculations are performed, this turns out to be preferable to attempting to open the door, which is the optimal action (see Figure 2(c)).

5.2 Locally-uniform abstraction

The ostrich effect occurs when states of different abstraction are considered, the problem being, as noted above, that the DP given by the abstraction is non-Markovian. The solution is to make the abstraction locally uniform, and therefore locally Markovian. In this way, the iterative step of the policy generation algorithm does not have to work across the edge of an abstract region, and, since the same information is available in all the states being considered at each point, there is no impetus for any of them to be favoured or avoided on that basis.

Specifically, let $O(e)$ denote the set of all possible outcomes of all actions in the state e , i.e. $O(e) = \{e' \in E : (\exists a \in A) \Pr(e, a, e') > 0\}$. For example, for Figure 3 (listing only x and $d3$ for each state and omitting outcome states not shown in Figure 3):

e	$O(e)$	comment
3, closed	{3, closed; 4, closed; 2, ., etc}	one outcome is abstract in $d3$
4, closed	{4, closed; 3, closed; 4, open; etc}	all at equal level of abstractness

Then, when calculating $\pi(e)$ for a particular e , if any member of $O(e)$ is abstract in a dimension in which e is concrete, e and all members of $O(e)$ are temporarily abstracted in that dimension. Note that E itself is not changed, and that different abstracting will occur to the same states depending on the e that is currently being planned for. Values for this local additional abstraction, \bar{V} , are the averages of the original V weighted by the relative *a priori* probabilities of the new and original states. These are used in the π update formula instead of the original V s. The values V are calculated from T , π and R in the usual fashion.

Continuing with the example above, consider first the state with $x = 3$, $d3 = \text{closed}$. Since the corresponding O contains the state with $x = 2$, $d3 = .$ (i.e., in which $d3$ is abstract), all members of O are reduced to $d3 = .$ The resulting values \bar{V} , calculated from the V , are shown in Figure 4 between the

states they correspond to (note that the values V are different from those in Figure 3, since the policy is different). The correct action, East (\rightarrow), is selected. On the other hand, consider the state with $x = 4$ and $d3 = \text{closed}$. Since the whole corresponding O is already at the same level of abstractness (the state with $x = 5$, $d3 = \cdot$ cannot be reached as it is on the other side of the closed door), no reduction is necessary. The action selected, Open (\circ), is again correct.

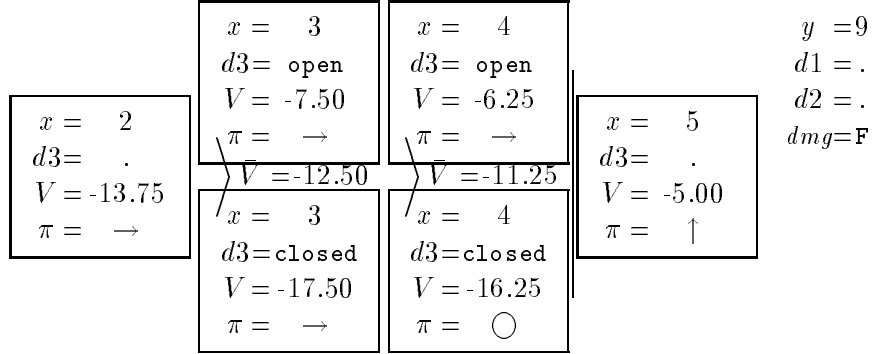


Figure 4: Locally-uniform abstraction

5.3 Effects and consequences

As can be seen from the above description, the door is only partially considered at $x = 3$. This means that, in most cases, the more concrete region must extend one step beyond the region in which the dimension is immediately relevant. For a dimension to be fully considered at a state, the possible outcomes of all actions at that state must also be concrete in that dimension.

Another, less obvious problem arises when the concrete region is insufficient. In some cases, the policy does not converge, instead oscillating between two possibilities. One must be careful, therefore, with the envelope to avoid these situations, or else to detect them and modify the envelope accordingly.

The oscillation occurs as follows. Consider the situation where the value of a state, say s_a , is averaged with another state, s_b . If that s_b is better than s_a , then actions (in s_a) which have a high probability of failing (or moving to s_b) seem somewhat more attractive than they should. However, once the action is chosen, the values are calculated correctly and the value of the state using that action is discovered to be lower than previously computed. Since now the value of s_a is much worse, so is the combined value, and actions which have a high probability of failing will now seem *less* attractive than they should.

Note that this is not necessarily fatal: in some situations, both resulting policies are workable, as is their alternation. They are sub-optimal, though, sometimes obviously so: in one of the policies, in the state immediately south of $d1$ the agent would try to open $d1$ if it was `closed`, but walk away from it and go via the alternative route when $d1$ was `open`.

To avoid such oscillation altogether, the envelope selection and modification algorithms need to ensure that the envelope is sufficient. At present, it appears that the algorithm described in Section 6.2 below solves the situation as it

arises; in the future, as the envelope modification algorithms are changed and improved, this problem will need to be watched for and corrected as it recurs.

6 Dynamic envelope abstraction

At the beginning of planning, an initial abstracted envelope E must be selected; as noted earlier, how this is done is crucial. As planning and execution progress, the envelope abstraction may need to be changed. Changes occur both as the planner gains a better idea of the best path to the goal, and as the agent itself moves from one part of the state space to another.

The two types of change are: (1) refinement, to make states more concrete, and (2) coarsening, to make states more abstract. These must be balanced, so that the planner makes the best possible use of the available computational resources. If too much refinement occurs, $|E|$ becomes too large and the planner will slow down, while when $|E|$ is too small the planner may not use all resources available.

The general strategy of our approach is that the envelope should be mostly concrete near the agent and its planned path to the goal, to allow detailed planning, but mostly abstract elsewhere, to conserve computational resources.

This section presents the initial envelope selection first, followed by two possible methods of envelope modification: one based on the policy calculated, the other on the likelihood of encountering particular states in the future, given the current state.

6.1 Selecting the initial envelope abstraction

In this section we describe a method for selecting the initial envelope based on the structured transition function and the reward function. As we have discussed earlier, real-world domains are too large for the transition function to be specified exhaustively, so a higher-level description will always be available. This description can therefore be used to construct the initial envelope by inferring the nexus between the dimensions (that is, linking points, those points at which the dimensions interact). Intuitively, the structure of the solution is likely to resemble the structure of the problem. The incorporation of the reward function reflects the assumption that the dimensions on which the reward is based will be important.

The two steps in deriving the initial envelope from the high-level domain descriptions are as follows. Firstly, all the dimensions mentioned in the reward tree are to be concrete throughout the envelope. In the sample domain, these are the x and y coordinates and the dmg dimension. Secondly, for each leaf node, the states matching the corresponding pre-state are to be concrete in all the dimensions mentioned in the ancestors of that leaf node. We call this a *nexus* between dimensions.

Some nexus are subsumed by the reward tree, and so will already be included in the envelope as concrete states and hence are not significant. In the sample domain, the significant nexus are on both sides of each door, in the two locations

immediately adjacent. This makes a total of 12 significant nexus, derived from 3 doors \times 2 sides \times 2 positions of that door (`open` / `closed`).

After the reward dimensions are included $|E| = 10 \times 10 \times 2 = 200$, and after the nexus are taken into account $|E| = 212$ (compared to $|S| = 1600$ specific states).

Note that when this is reduced to the locally-uniform abstraction form, using the algorithm described in Section 5.2, the more concrete states will be taken into account only to a very minimal extent; however, this is sufficient to trigger the policy-based envelope refinement algorithm below where necessary.

6.2 Policy-based Envelope Refinement

Consider two (or more) envelope states which differ only in the value of a single dimension and have a different policy. Suppose also that there is an adjacent state which is abstract in that dimension. In this case, the dimension in which they differ, and in which the adjacent state is abstract, will be made locally abstract during the policy generation algorithm. The difference in the policy of the two states indicates that the dimension is important at those two states, yet by the reduction to the locally-uniform abstraction form, undertaken to avoid the ostrich effect, it will not be taken into account fully. This suggests a test for refining the envelope: whenever there are three states c_1 , c_2 and a in the envelope such that $\pi(c_1) \neq \pi(c_2)$, $a \in O(c_1)$, and c_1 and c_2 differ in the values for a concrete dimension d in which a is abstract, but not in any dimensions in which a is concrete, add the dimension d to state a .

This algorithm, together with the one presented in 6.1 above, is how E was obtained for Section 5. It increases the number of envelope states from the initial 212 to 225.

In some cases, however, this policy-based envelope refinement propagates beyond the immediate vicinity: in the case of $d1$, where, in the states immediately south of the wall to the west of the door, the best action depends on whether the door is open or closed (see Figure 5); this difference does not diminish with distance: when $d1$ is `open`, the planner selects the upper path, while when it is `closed` it chooses the lower path. While indeed the best action does depend on $d1$, the difference in expected reward is not great, and in any case the best action depends equally on $d2$ and $d3$ which are abstract at that point. The proximity of the western edge of the world hides the extent of the problem: without it, the refinement would propagate westward without bound. This can be easily demonstrated by moving $d1$ eastwards two units. Again, the refinement propagates westward, until it meets the western edge (now two steps further from $d1$) or, theoretically, some other obstruction or perturbation. One way to solve this problem is to combine this refinement test with the envelope modification strategy described in next section.

6.3 Likelihood-based envelope modification

We noted above that the envelope should be mostly concrete near the agent and its planned path to the goal, to allow detailed planning, but mostly abstract

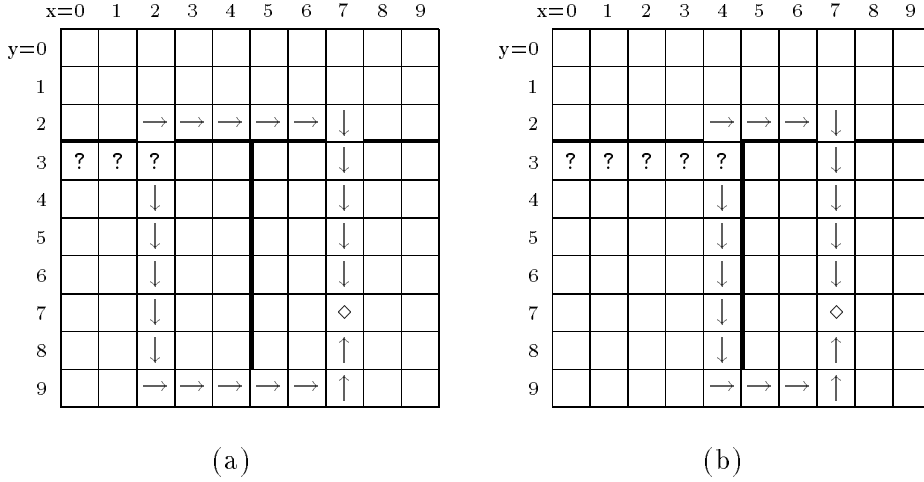


Figure 5: (a) Two paths to the goal; in the states marked ?, which path is optimal depends on $d1$ ($d2$ and $d3$ being equal). The envelope states corresponding to these locations will be made concrete in $d1$ by the policy-based refinement. (b) With $d1$ moved eastward, the policy-based refinement still propagates to the wall.

elsewhere, to conserve computational resources. Likelihoods are one way to realise this concept; they can be described as ‘discounted sum of probabilities’. The likelihood of a state decreases both as the state is further in the future and as it is less probable.

6.3.1 Likelihood Calculation

The formula for the likelihood l is similar to the formula for value, except the reward for the state is replaced with an ‘is current state’ function, cur , and the transition probabilities are reversed. That is,

$$l(s) \leftarrow cur(s) + \gamma_l \sum_{s'} \Pr(s', \pi(s'), s) l(s')$$

where γ_l is the likelihood discounting factor ($0 \leq \gamma_l < 1$) and

$$cur(s) = \begin{cases} 1 - \gamma_l & \text{if } s \text{ is the current state} \\ 0 & \text{otherwise} \end{cases}$$

This can also be expressed and solved as a set of linear equations; in matrix notation, this is

$$(I - \gamma_l T_\pi^T) l = cur$$

where T_π is the transition matrix induced by the policy π . The formula can be equally applied to the variously-abstract envelope: s is simply replaced by e in the above formula. For many purposes, however, the likelihoods will then need to be inversely scaled by the *a priori* probabilities to compensate for variation due to the varying abstractness.

To take into account further planning, and ensure that nearby absorbing states are kept in mind even if the current policy deterministically avoids them,

the formula is modified to use an estimated future policy $\hat{\pi}$ instead of π . In this estimate, $\hat{\pi}(s)$ is simply a distribution over actions which assigns some constant probability to the current $\pi(s)$ and distributes the remaining probability mass among the other actions equally. This distributed probability mass corresponds to the probability that the policy will change sometime in the future. The equation solved is

$$(I - \gamma_l T_{\hat{\pi}}^T)l = \text{cur}$$

There are two parameters for the likelihood calculation, the discounting factor γ_l and the probability of policy change.

Likelihoods for the initial situation ($\langle 0, 0, \text{closed}, \text{closed}, \text{closed}, \text{F} \rangle$) and a possible situation later in the execution ($\langle 4, 2, \text{closed}, \text{closed}, \text{closed}, \text{F} \rangle$) are shown in Figure 6; larger symbols correspond to higher likelihood. The symbols are based on the logarithm of the likelihood since the likelihoods shown in this figure range from 2^{-37} to $2^{-1.4}$. Note in particular that the state immediately north of $d2$, $\langle 7, 2 \rangle$, has a somewhat bigger likelihood than its neighbours — the planner expects to spend about 10 time steps while opening the door, as opposed to 1.25 in the other states on the shortest path. For these figures, $\gamma_l = 0.95$ and the probability of policy change is 10%.

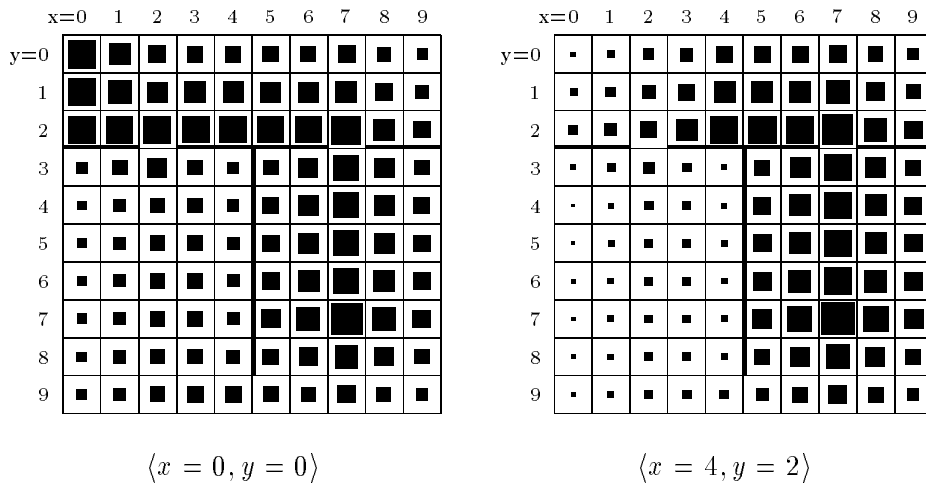


Figure 6: Likelihoods for the initial state, $\langle x = 0, y = 0 \rangle$, and a state later in the execution, $\langle x = 4, y = 2 \rangle$. The symbol size is based on the logarithm of the likelihood.

6.3.2 Features of likelihoods

One interesting feature of the resulting numbers is that they emphasise absorbing and near-absorbing states somewhat more than might be intuitively expected. However, considering that absorbing states are in general important, this is probably a good feature, especially since normally the planner will try to minimise the probability of entering an absorbing state (unless it is the goal). This feature should help ensure that absorbing states are kept in mind as long as there is any chance of falling into them. Dean *et al.* [7] note that such

undesirable absorbing states along the path to the goal tend to come up as candidates for removal from the envelope (due to the low probability of reaching them with the current policy), and have to make special accommodation for them so they are not removed from the envelope (done by making low value a criterion). With likelihood emphasising these states, such special handling is not necessary in our approach.

With the choice of $1 - \gamma_l$ as the constant for the current state, $\sum_s l(s) = 1$. This means that l can be interpreted as a distribution. In fact, if checked in the near future, the agent has a probability of $l(s)$ of being in state s , assuming it will follow the policy $\hat{\pi}$ and ‘near future’ is suitably defined (probability of checking at time t is $(1 - \gamma_l)\gamma_l^t$).

As noted above, concrete states are calculated as less likely on account of their concreteness. Informally, an abstract state will get all the likelihood for its corresponding more concrete states, whereas the concrete states compete with the other possibilities for the concrete dimensions. This bias in the likelihoods is exactly in line with the a priori probabilities of the envelope states; the *a priori* probabilities can be used to compensate for this.

Another measure of states likely to be encountered in the near future, used for example by [12] and [7], is the probability distribution after n time-steps, where n would ideally be the amount of time until the planner emits the next policy. As we have discussed above, this requires an additional test to distinguish between states that are genuinely far from the path to the goal (which should be pruned from the envelope), and those that have low probability of being encountered because they are undesirable (which should be kept in the envelope so the policy continues to avoid them). Since the planner has no performance guarantees, however, n needs to be estimated adaptively. Perhaps more importantly, it is not obvious that this is the best measure after all. For one thing, the next policy to be emitted should not only cover the state where the agent will be at the beginning of its use, but preferably all the states encountered until it is superseded, so in fact $2n$ should be used instead. More fundamentally, the planner should in any case have better foresight than one or even two planning cycles ahead. It would be good for the planner to work out the details of much of the route to the goal as early as possible, to avoid unpleasant surprises later, keeping in mind that the stochastic nature of the domain precludes definite planning.

7 Discussion and further work

The solution presented in this paper is based on selectively ignoring some of the dimensions in some parts of the state space. In other words, we obtain approximate solutions by varying the level of abstraction in different parts of the state space. This has strong implications, since the resulting approximation is no longer Markovian. However, the approach is both intuitively appealing and, as we have shown in the examples presented, appears to be a useful method of approximation. The robot only considers a door dimension when it is about to walk through the door; if it has to walk through a dozen doors, it can focus

on each door as it approaches, and forget about those already passed. We are currently looking at how to apply the approach to an interesting variation to the simple robot navigation problem, when a door is locked and the key is in another location; in this case, the robot has to consider the door dimension before it reaches it.

While the idea of non-uniform abstraction is a general one, we have described one possible representation, based on a decision tree structure, for both the transition function and the reward function. Obviously there is no single unique description of a particular domain; the same transition function can be described using different orderings of the conditions. As with ordinary decision trees, depending on the structure of the domain and the order of conditions, the overall size of the tree structure may vary enormously. Note that the derivation of the envelope state transitions from the full transition representation can be done automatically and off-line.

Other representations of the domain structure are possible, and the choice of representation may depend on the particular domain. For example, for some domains, it might be an advantage to use hierarchical directed graphs instead of tree structures, to reduce the repeated specification of subtrees by the domain constructor. Other possible representations include the decision tree representation of the conditional probabilities of a two-time slice Bayesian network [5, 3] and a probabilistic STRIPS representation [8, 13]. The particular representation will affect some aspects of the envelope selection and modification algorithms.

This paper deals with control error exclusively. Sensor error is not considered and it is assumed that the agent can accurately discern the current world state. The relaxation of the assumption of observability will require the investigation as to the potential of applying the idea of non-uniform abstraction used in this paper to work being done in the area of Partially Observable MDPs (POMDPs). It is also assumed that the agent accurately knows the state space, the reward function, and the transition function. Given a transition function representation based on decision trees, it may be possible to adapt existing techniques for learning decision trees [16].

The classical MDP policy generation algorithm when applied to a non-uniformly abstracted envelope, may result in a policy where the agent goes to particular states depending on whether it seems better to know or not to know. This is because the DP given by the abstraction is non-Markovian. We have described modifications to the policy generation algorithm, whereby we re-calculate values for nearby states based on a locally uniform abstraction for each state. This ensures that the resulting policy is not based on value of information, although it may still be suboptimal.

The most crucial part of the approach is envelope selection and modification. The quality of the initial envelope is important since it provides the basis for the modification algorithms. The policy-based refinement only extends the envelope, by adding dimensions to envelope states in particular situations. It requires a ‘seed’ of a more concrete area before it is triggered and only works along the edges of more concrete regions.

The current method makes all the dimensions mentioned in the reward tree

concrete throughout the initial envelope. In domains where the reward is a function of many (or all) dimensions, obviously this strategy would no longer be applicable and some sort of sensitivity analysis would need to be applied to determine which of the dimensions most influence the reward (which may vary across the state space).

We make the assumption that the agent continues computation as it is acting ('recurrent deliberation' of [7]). This means the agent does not need to plan so well for unlikely possibilities, and can therefore expend more of its planning on the most likely paths and on the closer future, expecting that when and if it reaches other parts of the state space, it can improve the approximation as appropriate. This is the basis for the likelihood-based envelope modification. The potential problem of too much propagation of the policy-based refinement is addressed when that method is combined with the likelihood calculation. This is because the states which are not near the shortest path are made abstract using the likelihood calculation, and states on the shortest path are made concrete. The interaction between the methods is currently being investigated.

Acknowledgements

The authors would like to thank David Albrecht for the suggestion of representing the problems as a FSM, i.e. considering E a grammar and using compiler-construction techniques. We would also like to thank anonymous reviewers for their comments on an earlier version of this paper.

References

- [1] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [2] Dimitri P. Bertsekas. *Dynamic Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1987.
- [3] Craig Boutilier. Correlated action effects in decision theoretic regression. In *UAI*, pages 30–37, 1997.
- [4] Craig Boutilier and Richard Dearden. Approximating value trees in structured dynamic programming. In *13th International Conference on Machine Learning*, pages 54–62, Bari, Italy, 1996.
- [5] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Exploiting structure in policy construction. In *IJCAI*, pages 1104–1111, Montreal, 1995.
- [6] Anthony R Cassandra, Leslie Pack Kaelbling, and James A Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. Technical Report TR CS-96-17, Computer Science, Brown University, 1996.

- [7] Thomas Dean, Leslie Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.
- [8] Richard Dearden and Craig Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.
- [9] Mark Drummond and John Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings AAAI-90*, pages 138–144. AAAI, 1990.
- [10] Robert P. Goldman, David J. Musliner, Kurt D. Krebsbach, and Mark S. Boddy. Dynamic abstraction planning. In *AAAI 97*, 1997.
- [11] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts, 1960.
- [12] Jak Kirman. *Predicting real-time planner performance by domain characterization*. PhD thesis, Brown University, 1994.
- [13] Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. In *Proceedings AAAI-94*. AAAI, 1994.
- [14] David J. Musliner, Edmund H. Durfee, and Kang G. Shin. World modeling for the dynamic construction of real-time plans. *Artificial Intelligence*, 74:83–127, 1995.
- [15] Ann Nicholson and Leslie Pack Kaelbling. Toward approximate planning in very large stochastic domains. In *Proceedings of the AAAI Spring Symposium on Decision Theoretic Planning*, 1994.
- [16] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California, 1993.
- [17] Nevin L Zhang and Weihong Zhang. Fast value iteration for goal-directed Markov Decision Processes. In *UAI*, 1997.