

# 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,ann}@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.

## 1 Introduction

For small to medium sized stochastic domains, the theory of Markov decision processes provides algorithms for generating the optimal plan [2, 3]. 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 [6, 4] to obtain an exact solution or [5, 8] an approximate solution by ignoring particular dimensions, however, their abstractions are fixed throughout execution. 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.

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 the distribution over the effect of its actions.

The remainder of this paper is organised as follows. In Sect.2 we briefly explain classical planning in MDPs and the structure of the agent, and present an example domain to facilitate explanation. Section 3 presents the idea of non-uniform abstraction, and discusses how it might be relatively simply represented. Section 4 deals with the modifications to classical MDP planning necessitated by this approximation, involving re-calculation of values based on a locally uniform abstraction. In Sect.5 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 Sect.6.

## 2 Planning in Stochastic Domains

A Markov Decision Process (MDP) is a tuple  $\langle S, A, T, R, s_0 \rangle$ , respectively the *state space*, *set of available actions*, *transition function*, *reward function* and *initial state*. The agent begins in state  $s_0 \in S$ . At time  $n$ , the agent selects an action  $a \in A$ , which, together with the current state  $s_n$  and  $T$  gives a distribution over  $S$  for  $s_{n+1}$ , the current state at time  $n+1$ . It also receives a reward of  $R(s_n)$ ; the agent aims to maximise the expected discounted sum of these rewards. It is well-known that in a fully-observable MDP, where the agent knows  $T$ ,  $R$  and  $s_n$  when selecting an action, 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  $\pi$ .

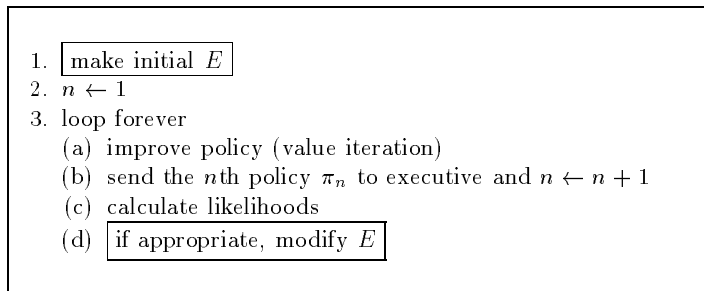
The standard algorithms for computing the policy are policy iteration [11] and value iteration [2]. Each consists of two iterative interrelated parts, one dealing with changing the policy, the other with computing the values. The iterative steps are:

$$\pi(s) \leftarrow \arg \max_a \sum_{s'} \Pr(s, a, s') V(s') \quad (1)$$

and

$$V(s) \leftarrow R(s) + \gamma \sum_{s'} \Pr(s, \pi(s'), s') V(s') \quad (2)$$

where  $\gamma$  the discounting factor is a parameter ( $0 \leq \gamma < 1$ ), the probabilities are specified by  $T$ , and  $V : S \rightarrow \mathfrak{R}$  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 this paper we use the value iteration algorithm, in which (1) and (2) are iterated together.



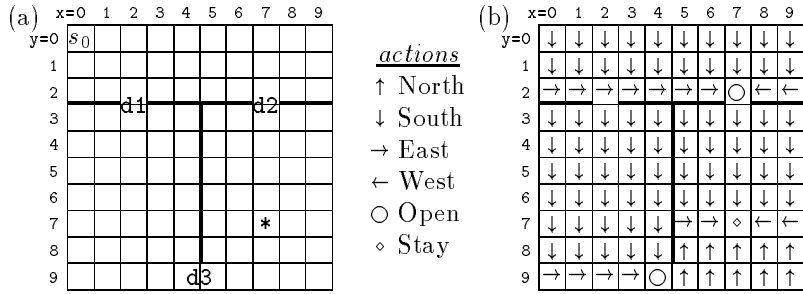
**Fig. 1.** Planner algorithm

As  $S$  becomes larger, the calculation of the optimal policy becomes more computationally expensive. If the state space is represented as a cartesian product of dimensions, each of which is a set,  $|S|$  will be exponential in the number of dimensions. To obtain at least an approximate solution, one can approximate the MDP with a MDP with a smaller state space  $E$ . For example, Dean *et al.* [7] use a subset of  $S$ , based on the notion of locality, and plans only for those states that are likely to be reached. If the state space has 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. One possibility is to make the reduced state space (or *envelope*)  $E$  the cartesian product of a subset of these dimensions [8, 14]. Excluding a dimension from  $E$  in this way is called ‘uniform abstraction’ or simply ‘abstraction’. It is also possible to exclude different dimensions from different parts  $E$  — this is called ‘non-uniform abstraction’, and is the focus of the research presented in this paper.

We assume an agent which has processing power available while it is acting. Architecturally, the agent is split into a *planner* and an *executive*; the planner continually improves policy, modifies the approximation and updates the focus based on the current state, while the executive selects actions based on the current state and current policy. This means that 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. Dean *et al.* [7] call this *recurrent deliberation*, and use it with their locality-based approximation. A similar architecture is used by the CIRCA system [13, 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 Fig.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. The other parts of the algorithm require only minor changes.

**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 Fig.2(a)). It is a  $10 \times 10$  grid, with one wall between



**Fig. 2.** Example domain. (a) Initial situation with starting location  $s_0$ , goal  $*$ , all three doors ( $d_1$ ,  $d_2$  and  $d_3$ ) closed and no damage; (b) Part of an optimal policy obtained using value iteration on the complete state space, with  $\gamma = 0.99999$

$y = 2$  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 indicates 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. More formally,  $S = S_x \times S_y \times S_{d_1} \times S_{d_2} \times S_{d_3} \times S_{dmg}$  where  $S_x = S_y = \{0, 1, \dots, 9\}$ ,  $S_{d_1} = S_{d_2} = S_{d_3} = \{\text{open, closed}\}$  and  $S_{dmg} = \{\text{T, F}\}$ . 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, that is,  $\langle 0, 0, \text{closed, closed, closed, F} \rangle$ .

There are six actions in  $A$ : movements in all four cardinal directions, `OpenDoor`, which opens the immediately adjacent door, and `Stay`, which is a do-nothing action. Each action may fail with some specified probability. The reward function  $R$  corresponds to a goal of achieving a particular location while avoiding damage. It gives 0 for a goal state ( $x = 7, y = 7, dmg = F$ ),  $-2$  for damage and  $-1$  for all other states. The reward necessary to force the agent to definitely avoid damage, corresponding to a CIRCA [13] control-level goal, depends on the probabilities. However, since the intended domains of application do not have guaranteed actions, guaranteed control-level goal satisfaction will usually not be possible anyway. We use discounting factor  $\gamma = 0.99999$  in this paper. An optimal policy for the grid locations with all doors closed and no damage is shown in Fig.2(b).

### 3 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. 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.

**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 dimensions. In the example domain, each  $e$  is  $e_x \times e_y \times e_{d1} \times e_{d2} \times e_{d3} \times e_{dmg}$ .

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  $S$ .

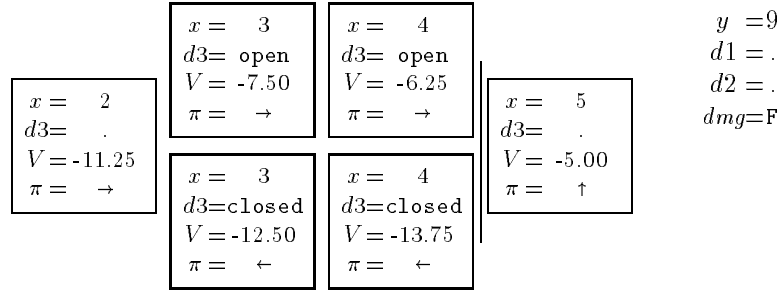
For example the envelope state having  $x = 7$ ,  $y = 2$ ,  $d2 = \text{closed}$ ,  $dmg = \text{F}$  and ignoring the dimensions  $d1$  and  $d3$ , would be represented as  $\{7\} \times \{2\} \times S_{d1} \times \{\text{closed}\} \times S_{d3} \times \{\text{F}\}$  and groups together four specific states, corresponding to the four combinations of `open` and `closed` for  $d1$  and  $d3$ . For brevity, we write  $\langle 7, 2, \cdot, \text{closed}, \cdot, \text{F} \rangle$ , dispensing with the set notation and using a period to denote an ignored dimension. 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.

To quantify the abstractness of a state, we use its a priori probability. This is an overall distribution over the states in the state space, without regard to the initial state or the agent’s policy. We specify the a priori probability of a specific state,  $\text{Pr}(s)$ , as 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. In the example, the a priori probabilities are uniform, so  $\text{Pr}(s)$  is the same for all  $s$  and  $\text{Pr}(e) = |e| \text{Pr}(s)$ .

**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, the transition function probabilities for each action are written in the form of a decision tree [15], enriched to allow non-determinism by allowing some nodes to decide at random rather than based on a particular dimension. The reward function  $R$ , is specified as an ordinary decision tree. No non-determinism is required, even for the abstracted state space, since the leaves can simply be combined linearly (averaged).

## 4 Policy Generation

Given an envelope of non-uniformly abstracted states, we can compute a policy using the standard value iteration (or policy iteration) algorithm described in Sect. 2. Note that this section pre-supposes the envelope; we discuss how to go about devising this envelope automatically in Sect. 5. The policy  $\pi$  becomes a



**Fig. 3.** Ostrich effect

mapping  $E \rightarrow A$  and the value function becomes  $V : E \rightarrow \mathfrak{R}$ . Equations (1) and (2) then read:

$$\pi(\epsilon) \leftarrow \arg \max_a \sum_{e'} \Pr(e, a, e') V(e') \quad (3)$$

and

$$V(\epsilon) \leftarrow R(\epsilon) + \gamma \sum_{e'} \Pr(e, \pi(e'), e') V(e') \quad (4)$$

where  $R(\epsilon)$  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 is translated back into  $S \rightarrow A$  form by setting  $\pi(s) = \pi(\epsilon)$  whenever  $s \in \epsilon$ . 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.

**Ostrich Effect.** When the agent plans to move from a relatively concrete state  $e_1$  to a more abstract state  $e_2$  and thence to  $e_3$ , it may need to know that  $e_2$  was reached from  $e_1$  in order to identify the correct  $e_3$ , which violates the Markov property. If the abstracted approximation is simply treated as a MDP in which the agent doesn't know which  $e_3$  it will reach, it won't correspond to the underlying process, which might reach a particular  $e_3$  deterministically. The problem is especially obvious when the actual  $e_3$  that will be reached is in fact  $e_1$  — the planner will plan a loop.

For a specific example of this, consider Fig.3 which 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  $\pi$  for that state. For clarity, the other dimensions ( $y = 9$ ,  $d1 = .$ ,  $d2 = .$ ,  $dmG = F$ ) 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  $x = 2$  ( $e_2$  in the general description) to  $x = 3$  ( $e_3$ ) 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.

**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.

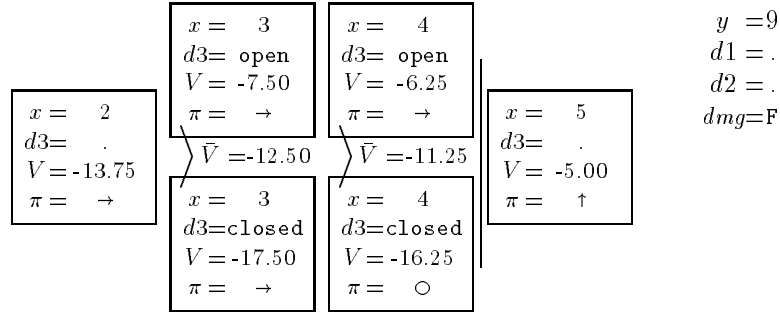
Specifically, let  $O(e)$  denote the set of all possible outcomes of all actions in the state  $e$ , i.e.  $O(e) = \{e' \in E : (\exists a \in A) \Pr(e, a, e') > 0\}$ . For example, for Fig.3 (listing only  $x$  and  $d3$  for each state and only states shown in Fig.3):

$e$	$O(e)$	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  $\pi$  update formula instead of the original  $V$ s. The values  $V$  are calculated from  $T$ ,  $\pi$  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 Fig.4 between the states they correspond to (note that the values  $V$  are different from those in Fig.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  $x = 5$ ,  $d3 = .$  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.

**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.



**Fig. 4.** Locally-uniform abstraction

At present, it appears that the algorithm described in Sect.5 below ensures the envelope is sufficient.

## 5 Dynamic Envelope Abstraction

Before 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 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. The planner must balance *refinement*, making states more concrete, and *coarsening*, making them more abstract, so that it makes the best possible use of the available 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.

**Selecting the Initial Envelope Abstraction** 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<sup>1</sup> 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

<sup>1</sup> Note that 'nexus' is both singular and plural.

in each action tree, 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. In the sample domain, nexus involving dimensions not already concrete are on both sides of each door, in the two locations immediately adjacent. This makes a total of 12 significant nexus, derived from  $3 \text{ doors} \times 2 \text{ sides} \times \{\text{open, closed}\}$ . 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). While the more concrete states will be taken into account only to a very minimal extent after reduction to the locally-uniform abstraction form (Sect.4), it is sufficient to trigger the policy-based envelope refinement algorithm below where necessary.

**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, and have an adjacent state which is abstract in that dimension. The difference in the policy of the two states indicates that the dimension is important there, 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 5 above, is how  $E$  was obtained for Sect.4. It increases the number of envelope states from the initial 212 to 225.

In some cases, this policy-based envelope refinement propagates beyond the immediate vicinity.<sup>2</sup> One way to solve this problem is to combine this refinement test with the envelope modification strategy described in next section.

**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 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. Unlike the measure used by [12] and [7], likelihoods are independent of planning cycle length.

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,

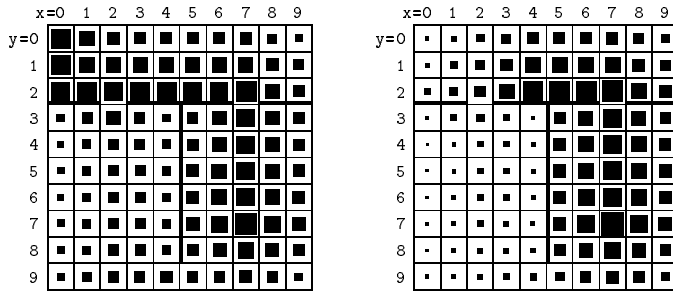
$$l(s) \leftarrow cur(s) + \gamma_l \sum_{s'} \Pr(s', \pi(s'), s) l(s') \quad (5)$$

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

$$cur(s) = \begin{cases} 1 - \gamma_l & \text{if } s \text{ is the current state} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

---

<sup>2</sup> Example described in [1].



**Fig. 5.** Likelihoods for the initial state,  $\langle x = 0, y = 0 \rangle$ , and a state later in the execution,  $\langle x = 4, y = 2 \rangle$ . Symbol size based on logarithm of likelihood.

This can be equally applied to the variously-abstract envelope:  $s$  is simply replaced by  $e$  in (5) and (6). 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, (5) is modified to use an estimated future policy  $\hat{\pi}$  instead of  $\pi$ . In this estimate,  $\hat{\pi}(e)$  is a distribution over actions which assigns some constant probability to the current  $\pi(e)$  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. This can also be expressed and solved as a set of linear equations; in matrix notation, the equation solved is  $(I - \gamma_l T_{\hat{\pi}}^T)l = \text{cur}$  where  $T_{\hat{\pi}}$  is the transition matrix induced by the policy  $\hat{\pi}$ . There are two parameters for the likelihood calculation, the discounting factor  $\gamma_l$  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 ( $\langle 4, 2, \text{closed}, \text{closed}, \text{closed}, \text{F} \rangle$ ), with  $\gamma_l = 0.95$  and probability of policy change 10%, are shown in Fig.5. Larger symbols correspond to higher likelihood; size is based on the logarithm of the likelihood since the likelihoods shown in the figure range from  $2^{-37}$  to  $2^{-1.4}$ .

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. This feature should help ensure that absorbing states are kept in mind as long as there is any chance of falling into them. With the measure of [12] and [7], absorbing states along the path to the goal tend to come up as candidates for removal from the envelope, as they are unlikely to be reached with the current policy, and special accommodation must be made for them so they are not removed from the envelope. 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,  $\sum_s l(s) = 1$ . In fact,  $l$  is the probability distribution for the current state of the agent in the

near future, provided it follows the policy  $\hat{\pi}$  and ‘near future’ is suitably defined (probability of checking at time  $t$  is  $(1 - \gamma_l)\gamma_l^t$ ).

## 6 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.

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 [15].

The classical MDP policy generation algorithm, 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.

We make the assumption that the agent continues computation as it is acting (‘recurrent deliberation’ of [7]). This means the agent can focus more closely on planning for the most likely paths and the closer future, expecting that it can

later improve the approximation for other parts of the state space as needed. 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.

## References

- [1] J. Baum and A. E. Nicholson. Dynamic non-uniform abstractions for approximate planning in large structured stochastic domains. Technical Report TR 98/18, Department of Computer Science, Monash Uni., 1998.
- [2] R. Bellman. *Dynamic Programming*. Princeton Uni. Press, 1957.
- [3] D. P. Bertsekas. *Dynamic Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1987.
- [4] C. Boutilier. Correlated action effects in decision theoretic regression. In *Proc. of UAI*, pages 30–37, 1997.
- [5] C. Boutilier and R. Dearden. Approximating value trees in structured dynamic programming. In *Proc. of 13th Int'l Conf. on Machine Learning*, pages 54–62, Bari, Italy, 1996.
- [6] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. In *Proc. of IJCAI*, pages 1104–1111, Montreal, 1995.
- [7] T. Dean, L. P. Kaelbling, J. Kirman, and A. E. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.
- [8] R. Dearden and C. Boutilier. Abstraction and approximate decision theoretic planning. *Artificial Intelligence*, 89(1):219–283, 1997.
- [9] M. Drummond and J. Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. of AAAI-90*, pages 138–144. AAAI, 1990.
- [10] R. P. Goldman, D. J. Musliner, K. D. Krebsbach, and M. S. Boddy. Dynamic abstraction planning. In *Proc. of AAAI 97*, 1997.
- [11] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachusetts, 1960.
- [12] J. Kirman. *Predicting real-time planner performance by domain characterization*. PhD thesis, Brown Uni., 1994.
- [13] D. J. Musliner, E. H. Durfee, and K. G. Shin. World modeling for the dynamic construction of real-time plans. *Artificial Intelligence*, 74:83–127, 1995.
- [14] A. E. Nicholson and L. P. Kaelbling. Toward approximate planning in very large stochastic domains. In *Proc. of Spring Symposium on Decision Theoretic Planning*, 1994.
- [15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, California, 1993.