PPCP: Efficient Probabilistic Planning with Clear Preferences in Partially-Known Environments *

Maxim Likhachev
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
maxim+@cs.cmu.edu

Anthony Stentz
The Robotics
Carnegie Mellon University
Pittsburgh, PA 15213
axs@rec.ri.cmu.edu

Abstract

For most real-world problems the agent operates in only *partially*-known environments. Probabilistic planners can reason over the missing information and produce plans that take into account the uncertainty about the environment. Unfortunately though, they can rarely scale up to the problems that are of interest in real-world. In this paper, however, we show that for a certain subset of problems we can develop a very efficient probabilistic planner. The proposed planner, called PPCP, is applicable to the problems for which it is clear what values of the missing information would result in the best plan. In other words, there exists a clear preference for the actual values of the missing information. For example, in the problem of robot navigation in partially-known environments it is always preferred to find out that an initially unknown location is traversable rather than not. The planner we propose exploits this property by using a series of deterministic A*-like searches to construct and refine a policy in anytime fashion. On the theoretical side, we show that once converged, the policy is guaranteed to be optimal under certain conditions. On the experimental side, we show the power of PPCP on the problem of robot navigation in partially-known terrains. The planner can scale up to very large environments with thousands of initially unknown locations. We believe that this is several orders of magnitude more unknowns than what the current probabilistic planners developed for the same problem can handle. Also, despite the fact that the problem we experimented on in general does not satisfy the conditions for the solution optimality, PPCP still produces the solutions that are nearly always optimal.

Introduction

For many real-world problems the agent operates in an environment that is only partially-known. Examples of such problems include robot navigation in partially-known terrains, route planning under partially-known traffic conditions, air traffic management with changing weather conditions and others. Ideally, in all of these situations, to produce a plan, a planner needs to reason over the probability distribution over all the possible instances of the environment. Such planning corresponds to POMDP planning, however, and is known to be hard (PSPACE-complete (Papadimitriou & Tsitsiklis 1987)).

For many of these problems, however, one can clearly name beforehand the best values of the variables that represent the unknowns in the environment. We call such values *preferred* values. Thus, in the robot navigation problem, for example, it is always preferred to find out that an initially unknown location is traversable rather than not. In the problem of route planning under partially-known traffic conditions, it is always preferred to find out that there is no traffic on the road of interest. And in the air traffic management problem it is always preferred to have a good flying weather. In this paper we give an algorithm called PPCP (Probabilistic Planning with Clear Preferences) that is able to scale up to very large problems by exploiting this property.

PPCP constructs and refines a plan by running a series of deterministic searches. By making a certain approximating assumption about the problem, PPCP keeps the complexity of each search low and independent of the amount of the missing information. Each search is extremely fast, and running a series of fast low-dimensional searches turns out to be much faster than solving the full problem at once since the memory requirements are much lower. While the assumption the algorithm makes does not need to hold for the found plan to be valid, it is guaranteed to be optimal if the assumption holds.

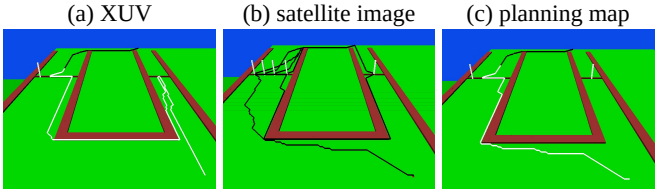
We demonstrated the power of PPCP on the problem of robot navigation in partially-known environments. We found that it was able to scale up to large (0.5km by 0.5km) environments with thousands of initially unknown locations.

The Motivating Problem

The motivation for our research was the problem of robot navigation in partially-known terrains. In outside environments it is often the case that the robot has only a rough map of the environment (for example, a map based on a satellite image as shown in figure 1(b), or a map constructed from previous scans of the environment). In such a map many places can not be classified as traversable or untraversable

*This work was sponsored by the U.S. Army Research Laboratory, under contract Robotics Collaborative Technology Alliance (contract number DAAD19-01-2-0012). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The authors would also like to thank Dave Ferguson for his very helpful comments on the paper.

(a) XUV (b) satellite image (c) planning map



(d) navigation with freespace assumption (e) policy produced by PPCP (f) navigation with PPCP planning

Figure 1: Robot Navigation in Partially-Known Terrain

with full certainty. Instead, for each such place we can estimate a probability of it being obstructed.

Figure 1(a) shows the ground vehicle used to motivate this task. The robot has a number of sensors on it including a laser rangefinder for scanning environment. Figure 1(b) shows a satellite image of the environment in which the robot operates. It covers the area of 1km by 1km area. This image is post-processed to produce a discretized 2D environment of size 1000 by 1000 cells. Each cell is associated with the cost of traversability. Figure 1(c) shows a simpler planning map of size 200 by 200 cells. In this particular example, 5 cells are assumed to be unknown (shown as raised white rectangles) and are associated with the probability of them being obstructed; we assume independence between unknown squares. To simplify the problem we only consider two possibilities for each of these cells: traversable or not.[1] (The described PPCP algorithm, however, can handle more than two possible values for each unknown cell.) The cells that are occupied by obstacles are shown in dark red color. The robot is located in the near right corner of the map, while its goal is at the far end of the map.

The robot must take the unknown locations into account when planning for its route. It may plan a path through these locations, but it risks having to turn back if its way is blocked. For example, figure 1(d) shows the path traversed by the robot if it always assumes that unknown locations are free unless sensed otherwise and plans and follows the shortest trajectories under this assumption (freespace assumption (Koenig & Smirnov 1996)). The trajectory traversed by the robot is shown in white, while the planned path is shown in black. We assume that the sensing range of the robot is 1 meter, which corresponds to 1 cell. We also assume perfect sensors: after the robot senses a square it knows its true state forever after.

In contrast to planning with freespace assumption, figure 1(e) shows the full policy after convergence (in black color) produced by our algorithm. This policy happens to be optimal: it minimizes the expected distance traveled by the robot before it reaches its destination. According to the policy the robot should first try the path on its left since there are 4 unknown locations in there and therefore there is a higher chance that one of them will be free and the robot will be able to go through. Figure 1(f) shows the actual trajectory traversed by the robot after it executed the policy.

Designing a good policy for the robot is a POMDP planning problem, which are in general very hard to solve. In the POMDP representation, a state is the position of the robot and the true status of each unknown location. The position of the robot is observable, and the hidden variables are whether each unknown place is occupied. The number of states is (# of robot locations)$\times 2^{\#\text{ of unknown places}}$. So, the number of states is exponential in the number of unknown places and therefore quickly becomes very large. We can define, however, clear preferences for the unknown locations: it is always preferred to have an unknown location to be free rather than obstructed. This makes our PPCP algorithm applicable to the problem.

Notations and Assumptions

In this section we introduce some notations and formalize mathematically the class of problems our algorithm is suitable for. Throughout the rest of the paper, to help understand various concepts we will often explain them on the problem of robot navigation in partially-known environments described in the previous section.

We assume that the environment is fully deterministic and can be modeled as a graph. That is, if we were to know the true value of each variable that represents the missing information about the environment then there would be no uncertainty in an outcome of any action. There are certain elements of the environment, however, whose status we are uncertain about and which affect the outcomes (and/or possible costs) of one or more actions. In the following we rephrase this mathematically.

Let X be a full state-vector (a belief state). We assume it can be split into two sets of variables, $S(X)$ and $H(X)$: $X = [S(X); H(X)]$. $S(X)$ is the set of variables whose values are always observed such as the position of the robot in the robot navigation problem. $H(X)$ is the set of (hidden) variables that initially represented the missing information about the environment. These correspond to the status of unknown cells in the robot navigation problem. We will use $h_i(X)$ to denote an i^{th} variable in $H(X)$. The variables in $H(X)$ are never moved to $S(X)$ and vice versa. We restrict the variables in $S(X)$ and $H(X)$ to be discrete and with a finite range of possible values. X_{start} is used to denote the start state: in the robot navigation problem, the value of $S(X_{\text{start}})$ is the start location of the robot, while the values of all the variables in $H(X_{\text{start}})$ are unknown. The goal of the planner is to construct a policy that reaches any state X such that $S(X) = S_{\text{goal}}$, where S_{goal} is given, while minimizing the expected cost of execution. In the robot navigation problem, S_{goal} corresponds to the goal location of the robot.

[1] This assumption also models better the problem of robot navigation in the environment where some locations can be occupied by adversaries. We are actually more interested in this problem, but use the problem of robot navigation in partially-known terrains to illustrate the algorithm since it is easier to describe, is highly related and is also an important problem in robotics.

We assume perfect sensing. For the sake of easier notation let us introduce an additional value u for each variable $h_i \in H$. The setting $h_i(X) = u$ at state X will represent the fact that the value of h_i is unknown at X. If $h_i(X) \neq u$, then the true value of h_i is known at X since sensing is perfect: if a cell has ever been sensed its status is known. In the robot navigation problem, we assume that the robot's sensors are noise free: it can sense the status of a neighboring cell exactly.

We assume at most one hidden variable per action. Let $A(S(X))$ represent the set of actions available at any state Y whose $S(Y) = S(X)$. (The actions are associated with $S(X)$ since the values of variables in $H(X)$ affect purely the costs and outcomes of actions.) Each action $a \in A(S(X))$ taken at state X may have one or more outcomes. If the execution of an action does not depend on any of the variables h_i whose values are not yet known, then there is only one outcome of a. Otherwise, there can be more than one outcome. We assume that each such action can not be controlled by more than one hidden variable. The value of one hidden variable can affect more than one action though. We use $h^{S(X),a}$ to represent the hidden variable that controls the outcomes and costs of action a taken at $S(X)$. By $h^{S(X),a} = \textbf{null}$ we denote the case when there was never any uncertainty about the outcome of action a taken at state X. In other words, none of the variables in H control the outcomes of a executed at $S(X)$, and since the underlying environment is deterministic, there is only one outcome.

The set of possible outcomes of action a taken at $S(X)$ is notated by $succ(S(X), a)$, whereas $c(S(X), a, S(Y))$ such that $S(Y) \in succ(S(X), a)$ denotes the cost of the action and the outcome $S(Y)$. The costs are assumed to be bounded from below by a (small) positive constant. Sometimes, we will need to refer to the set of successors in the belief state-space. In these cases we will use the notation $succ(X, a)$ to denote the set of belief states Y such that $S(Y) \in succ(S(X), a)$ and $H(Y)$ is the same as $H(X)$ except for $h^{(S(X),a)}(Y)$ which becomes known if it was unknown at X and remains the same otherwise. The function $P_{X,a}(Y)$, the probability distribution of outcomes of action a executed in state X, follows the probability distribution of $h^{S(X),a}$, $P(h^{S(X),a})$. Once action a was executed in state X the actual value of $h^{S(X),a}$ can be deduced since we assumed the sensing is perfect and the environment is deterministic.

In our robot navigation problem each unknown cell is represented by its own hidden variable. Whenever the robot senses a previously unobserved location, the value of the corresponding variable h_i transitions from u to either 0 or 1 with the appropriate probabilities and remains there afterwards.

We assume independence of the hidden variables. For the sake of efficient planning we assume that the variables in H can be considered independent of each other and therefore $P(H) = \prod_{i=1}^{|H|} P(h_i)$. In the robot navigation problem, the independence assumption corresponds to the assumption that the status of an unknown cell h_i is independent of the status of an unknown cell h_j, $i \neq j$. In reality, of course, they may depend on each other, especially if they are adjacent. Nevertheless, their independence is commonly assumed in order to ease the planning process.

We assume clear preferences on the values of the hidden variables are available. We require that for each variable $h_i \in H$ we are given its preferred value, denoted by b (i.e., best). This value must satisfy the following property. Given any state X and any action a such that $h^{S(X),a}$ is not known (that is, $h^{S(X),a}(X) = u$), there exists a successor state X' such that $h^{S(X),a}(X') = b$ and $X' = \text{argmin}_{Y \in succ(X,a)} c(S(X), a, S(Y)) + v^*(Y)$, where $v^*(Y)$ is the expected cost of executing an optimal policy at state Y (**Def. 1**). We will use the notation $succ(X, a)^b$ (i.e., the best successor) to denote the state X' whose $h^{S(X),a}(X') = b$ if $h^{S(X),a}(X) = u$ and whose $h^{S(X),a}(X') = h^{S(X),a}(X)$ otherwise. (In the latter case it may even be possible that $h^{S(X),a}(X') \neq b$. There were simply no other outcomes of action a executed at X since $h^{S(X),a}(X) \neq u$.) Our robot navigation problem clearly satisfies this property since for each sensing action there always exist two outcomes: a cell is blocked or unblocked. The latter is clearly the preferred outcome.

Algorithm

In the first section we present a planner that constructs a provably optimal policy using a series of A*-like searches in the belief state-space. Each search, however, can be very expensive since the size of the belief state-space is exponential in the number of hidden variables. In the following section we show how the planner can be extended to use a series of searches in the underlying environment instead.

Optimal Planning via Repeated Searches

The algorithm works by executing a series of deterministic searches. We will first describe how the deterministic search works and then we will show how the main function of the algorithm uses these searches to construct the first policy and then refine it.

The function that does the deterministic search is called ComputePath and is shown in figure 2. The search is done in the belief state-space. It is very similar to (backward) A*. It also computes g-values of states, uses heuristic values to focus its search and performs repeated expansions of states in the order of g-value plus heuristic (known as f-values). The meaning of the g-values and the criteria that the solution minimizes, however, is somewhat different from the ones in A* search. Suppose that for each state X we have an estimate $v(X)$ of $v^*(X)$, the minimum expected cost of reaching a goal from the state. These estimates are provided to the search by the main function of the algorithm (they are always non-negative). Every state-action pair X' and $a \in A(X')$ then has a $Q_{v,g}(X', a) > 0$ associated with it that is calculated from the action costs $c(S(X'), a, S(Y))$ and value estimates $v(Y)$ for all states $Y \in succ(X', a)$ and the g-value for state $Y^b = succ(X', a)^b$. $Q_{v,g}(X', a)$ is defined as follows:

$$Q_{v,g}(X', a) = \sum_{Y \in succ(X',a)} P_{X',a}(Y) \cdot \max(c(S(X'), a, S(Y)) + v(Y), c(S(X'), a, S(Y^b)) + g(Y^b)) \quad (1)$$

Suppose for a moment that the provided v-values are equal to the corresponding v^*-values and $g(Y^b)$ is also equal to $v^*(Y^b)$. Since $Y^b = succ(X', a)^b$ is the preferred outcome of action a, according to the definition 1, $c(S(X'), a, S(Y)) + v(Y) \geq c(S(X'), a, S(Y^b)) + g(Y^b)$ for any outcome $Y \in succ(X', a)$. As a result, the equation 1 corresponds to the standard definition of an undiscounted Q-value (in terms of costs, rather than rewards though): the expectation over the sum of the immediate cost plus the value of an outcome. The plan that has the minimum expected cost of reaching the goal would then be given by a simple strategy of always picking an action with the smallest $Q_{v,g}(X, a)$ at any current state X.

In reality, v-values may not necessarily be equal to the corresponding v^*-values at first. Nevertheless, the search computes g-values based on $Q_{v,g}$-values. In particular, suppose g^*-values are the solution to the following fixpoint equation:

$$g^*(X) = \begin{cases} 0 & \text{if } S(X) = S_{\text{goal}} \\ \min_{a \in A(S(X))} Q_{v,g^*}(X, a) & \text{otherwise} \end{cases} \quad (2)$$

Then, setting v-values to be equal to the corresponding v^*-values, would also make $g^*(X) = v^*(X)$ for every state X. Also, because of the max operator in the equation 1 and the fact that all costs are positive, $Q_{v,g^*}(X', a)$ is always strictly larger than $g^*(succ(X', a)^b)$, independently of whether v-values are correct estimates or not. Thus, the trajectory from any state with a finite g^*-value to a goal state is guaranteed to be acyclic and reach the goal, when constructed by always picking an action $a_{\min} = \text{argmin}_{a \in A(S(X))} Q_{v,g^*}(X, a)$ at any state X and then moving into the state $succ(X, a_{\min})^b$. This acyclicity allows us to perform a deterministic search which computes and sets the g-values of relevant states to their corresponding g^*-values.

The ComputePath function, shown in figure 2, searches backwards in the belief state-space from goal states towards the state X_{p} on which it was called. (It is important to remember that the number of goal states in the belief state-space is exponential in the number of hidden variables, since a goal state is any state X whose $S(X) = S_{\text{goal}}$.) The trajectory the search returns uses only the transitions that correspond to either deterministic actions or preferred outcomes of stochastic actions. This is implemented by starting off the search with all and only those goal states, whose hidden variables assume unknown or preferred values if they are also unknown in $H(X_{\text{p}})$ and values equal to the corresponding hidden variables in $H(X_{\text{p}})$ otherwise (lines 3– 6). The first time ComputePath is called, X_{p} is X_{start} and therefore all the hidden variables are unknown. For the subsequent calls, however, X_{p} can be a different state, and the values of some of its hidden variables can be known.

Just like (backward) A* search, the ComputePath function expands states in the order of g plus heuristic and during each expansion (lines 8– 13) updates the g-value and $besta$ pointer of each predecessor state of the expanded state. It differs from A* though in that g-values are computed according to formula 1. Heuristics are used to focus the search. Since the search is backward, they estimate the cost of following a least-cost trajectory from X_{p} to state in question. In the pseudocode, a user-provided

```
1  procedure ComputePath(X_p)
2  g(X_p) = ∞; OPEN = ∅;
3  for every H whose every element h_i satisfies:
     [(h_i = u ⋁ h_i = b) ⋀ h_i(X_p) = u] OR [h_i = h_i(X_p) ⋀ h_i(X_p) ≠ u]
4    X = [S_goal; H];
5    g(X) = 0, besta(X) = null;
6    insert X into OPEN with g(X) + heur(X_p, X);
7  while(g(X_p) > min_{X'∈OPEN} g(X') + heur(X_p, X'))
8    remove X with smallest g(X) + heur(X_p, X) from OPEN;
9    for each action a and state X' s.t. X ∈ succ(X', a)
10     compute Q_{v,g}(X', a) according to formula 1;
11     if this search hasn't seen X' yet or g(X') > Q_{v,g}(X', a)
12       g(X') = Q_{v,g}(X', a); besta(X') = a;
13       insert/update X' in OPEN with the priority g(X') + heur(X_p, X');
14 procedure UpdateMDP(X_pivot)
15 X = X_pivot;
16 while (S(X) ≠ S_goal)
17   v(X) = g(X);
18   X = succ(X, besta(X))^b;
19 procedure Main()
20 X_pivot = X_start;
21 while (X_pivot! = null)
22   ComputePath(X_pivot);
23   UpdateMDP(X_pivot);
24   find state X on the current policy such that S(X) ≠ S_goal and it has
       v(X) < E_{X'∈succ(X,besta(X))} c(S(X), besta(X), S(X')) + v(X');
25   if found set X_pivot to X;
26   otherwise set X_pivot to null;
```

Figure 2: Optimal planning via repeated searches

function $heur(X_{\text{p}}, X)$ returns a heuristic value for state X. These values need to be consistent in the following sense: $heur(X_{\text{p}}, X_{\text{p}}) = 0$ and for every other state X and action $a \in A(S(X))$ $heur(X_{\text{p}}, succ(X, a)^b) \leq heur(X_{\text{p}}, X) + c(S(X), a, S(succ(X, a)^b))$. This reduces to normal consistency requirement on heuristics (Nilsson 1971) if the state-space is fully deterministic, that is, no information is missing at the time of planning. As an example, in the robot navigation problem the heuristic $heur(X_{\text{p}}, X)$ can be an estimate of the distance from $S(X_{\text{p}})$ to $S(X)$ under the assumption that all cells are free. Alternatively, a more informative but much harder to compute heuristic can assume that only all unknown cells are free.

The search finishes as soon as the g-value of X_{p} is no larger than the smallest priority of states in *OPEN*. (A min operator over empty set is assumed to return infinity. The same assumption was made about the expectation operator on line 24.) Once the ComputePath function exits the following holds for the path from X_{p} to a goal state constructed by always picking action $besta(X)$ at any state X and then moving into the state $succ(X, besta(X))^b$ if $besta(X)$ has more than one outcome: the g-value of every state on the trajectory is equal to the g^*-value of the same state.

The Main function of the algorithm (shown in figure 2) uses this fact by repeatedly executing searches on states that reside on the current policy (defined by $besta$ pointers) and whose v-values are smaller than what they should be according to the v-values of the successors of the policy action (line 24). The initial v-values need to be smaller than or equal to the costs of least-cost trajectories to a goal under the assumption that all hidden variables are equal to b (a simple initialization to zero suffices). After each search, UpdateMDP function updates the v-values of the states on the

path found by the search by setting them to their g-values, which, as mentioned above, are equal to their corresponding g^*-values. On one hand, this increases v-values and is guaranteed to correct the error between the v-values of these states and the v-values of their successors in the policy. On the other hand, g^*-values are bounded from above by v^*-values as long as v-values do not overestimate v^*-values. As a result, the algorithm converges, and at that time, the states on the found policy have their v-values equal to their v^*-values and the found policy itself is optimal (the proof of this and other theorems can be found in (Likhachev & Stentz 2003)):

Theorem 1 *The Main function in figure 2 terminates and at that time the expected cost of the policy defined by $besta$ pointers is given by $v(X_{\rm start})$ and is equal to the minimum expected cost of reaching a goal from $X_{\rm start}$.*

Scaling Up Searches

Each search in the version of the algorithm just presented can be very slow because it operates in the belief state-space whose size is exponential in the number of hidden variables. We now describe the final version of PPCP that addresses this inefficiency. At a high level, the main idea is to perform each search in the underlying environment rather than the full belief state-space. In other words, a search state consists of $S(X)$ variables only and the size of the search state-space is therefore independent of the amount of missing information.

The following assumption allows us to do this. Suppose the agent executes some action a whose outcome it is uncertain about at a belief state X. Suppose also that the execution puts the agent into $succ(X,a)^b$. This means that the agent can deduce the fact that the value of the hidden variable $h^{S(X),a}$ that represents the missing information about action a is b. During each search, however, we assume that in the future the agent will not need to execute action a or any other action whose outcome is dependent on the value of this hidden variable (remember that the value of a single hidden variable is allowed to control the outcomes of more than one action). In case it does need to execute such action again, the search assumes that the value of the hidden variable is unknown again. As a result, the search does not need to remember whether $h^{S(X),a}$ is unknown or known to be equal to b. In fact, whenever the search needs to query a v-value of any non-preferred outcome of any stochastic action in order to compute $Q_{v,g}$-value, it assumes that the values of all hidden variables are unknown unless they were known to have non-preferred values in the belief state $X_{\rm p}$, the state the ComputePath function was called on. Under this assumption, the calculation of $Q_{v,g}(X)$-value (equation 1) becomes independent of $H(X)$ since any hidden variable in $H(X)$ whose value is different from the same variable in $H(X_{\rm p})$ can only be equal to b, and these are replaced by u. Thus, each search can be done in the actual underlying environment rather than the exponentially larger belief state-space. The search no longer needs to keep track of $H(\cdot)$ part of the states, it operates directly on $S(\cdot)$ states.

The ComputePath function, shown in figure 3 performs this search. While the function is called on a belief state $X_{\rm p}$,

```
1  procedure ComputePath(X_p)
2  g(S(X_p)) = ∞; OPEN = ∅;
3  g(S_goal) = 0, besta(S_goal) = null;
4  insert S_goal into OPEN with g(S_goal) + heur(S(X_p), S_goal);
5  while(g(S(X_p)) > min_{S(X)∈OPEN} g(S(X)) + heur(S(X_p), S(X)))
6    remove S(X) with smallest g(S(X)) + heur(S(X_p), S(X)) from OPEN;
7    for each action a and S(X') s.t. S(X) = S(succ([S(X'); H^u(X_p)], a)^b)
8      compute Q̃_{v,g}(S(X'), a) according to formula 3;
9      if this search hasn't seen S(X') yet or g(S(X')) > Q̃_{v,g}(S(X'), a)
10       g(S(X')) = Q̃_{v,g}(S(X'), a); besta(S(X')) = a;
11       insert/update S(X') in OPEN with the priority
                                          g(S(X')) + heur(S(X_p), S(X'));
12 procedure UpdateMDP(X_pivot)
13 X = X_pivot;
14 while (S(X) ≠ S_goal)
15   v(X) = g(S(X)); v(X^u) = g(S(X)); besta(X) = besta(S(X));
16   X = succ(X, besta(X))^b;
17 procedure Main()
18 X_pivot = X_start;
19 while (X_pivot != null)
20   ComputePath(X_pivot);
21   UpdateMDP(X_pivot);
22   find state X on the current policy such that S(X) ≠ S_goal and it has
       v(X) < E_{X'∈succ(X,besta(X))} c(S(X), besta(X), S(X')) + v(X');
23   if found set X_pivot to X;
24   otherwise set X_pivot to null;
```

Figure 3: PPCP: Planning via repeated efficient searches

it searches backwards from a single state, $S_{\rm goal}$, towards a single state $S(X_{\rm p})$, and the states in the search state-space consist only of variables in S. The search assumes that each action a has only one outcome. It assumes that $S(X)$ is an outcome of action a executed at $S(X')$ if and only if $S(X) = S(succ([S(X'); H^u(X_{\rm p})], a)^b)$ (line 7), where $H^u(X_{\rm p})$ is defined as $H(X_{\rm p})$ but with each hidden variable equal to b set to u. This corresponds to setting up a deterministic environment in which each action a executed at $S(X')$ has a single outcome corresponding to the value of the hidden variable $h^{S(X'),a}$ in $H(X_{\rm p})$ if it is known there and to the preferred value of $h^{S(X'),a}$ if it is unknown in $H(X_{\rm p})$. The heuristics need to satisfy normal consistency requirements (Nilsson 1971) with respect to this environment we have just set up. The ComputePath function performs backward A* search in this state-space with the exception of how g-values are computed.

To implement the assumption that the agent has no memory for the preferred values of the hidden variables that were previously observed we need to compute $Q_{v,g}$-values appropriately. For any state-action pair $(S(X), a)$, $a \in A(S(X))$ the ComputePath function now computes g-values based on $\tilde{Q}_{v,g}(S(X), a)$ instead. Let X^u denote a belief state $[S(X); H^u(X_{\rm p})]$. We then define $\tilde{Q}_{v,g}(S(X), a)$ as:

$$\tilde{Q}_{v,g}(S(X),a) = \begin{array}{l} Q_{v,g}(X^u,a), \\ \text{where } g(succ(X^u,a)^b) = g(S(succ(X^u,a)^b)) \end{array} \quad (3)$$

According to this formula during the computation of the Q-value of action a we assume that we execute this action at a belief state X^u, at which we are unaware of any hidden variables with preferred values. In the calculation of $Q_{v,g}(X^u,a)$ (eq. 1) we then use v-values of the corresponding belief states. The calculation of $Q_{v,g}(X^u,a)$ also requires the g-value of $succ(X^u,a)^b$. The search does not compute g-values for full belief states. Instead, $g(succ(X^u,a)^b)$ is substituted with $g(S(succ(X^u,a)^b))$.

This computation of $\tilde{Q}_{v,g}$ implements the assumption that the agent does not remember the values of the hidden variables whenever they are detected as b.

In the robot navigation problem this ComputePath function corresponds to searching from the robot's goal location towards the cell whose x, y position is given by $S(X_{\mathrm{p}})$. The search is done in a 2D gridworld that corresponds to the environment, where each unknown cell is assumed to be blocked if it is known to be such in $H(X_{\mathrm{p}})$ and is assumed to be free otherwise. This search differs from the backward version of A* search only in how g-values are computed.

The Main and UpdateMDP functions (figure 3) operate in the exact same way as before with the exception that for each state X the UpdateMDP function updates, it also needs to update the corresponding belief state X^u (line 15). Note that the UpdateMDP function updates an actual policy that can be executed. Therefore, the successors of $besta(X)$ depend on the value of the hidden variable $h^{S(X),a}$ in $H(X)$, which is not necessarily equal to the one used to set up the search environment or the value of $h^{S(X),a}$ in $H(X^u)$.

Theoretical Properties We now present several theorems about the algorithm. First, let us define v^u-values which are somewhat similar to v^*-values.

$$v^u(X) = \begin{cases} 0 & \text{if } S(X) = S_{\mathrm{goal}} \\ \min_{a \in A(S(X))} Q_{v^u, v^u}(X^u, a) & \text{otherwise} \end{cases}$$

$v^u(X)$ is the minimum expected cost of a policy under which the preferred values of hidden variables are forgotten as soon as they are observed with a small caveat. The preferred values of hidden variables must remain the same independently of whether we forget the preferred values of hidden variables or not.

The first two theorems show several basic properties of the algorithm.

Theorem 2 *Every time UpdateMDP exits, for each state X whose v-value was just updated by UpdateMDP function it holds that $v(X) = 0$ if $S(X) = S_{\mathrm{goal}}$ and $v(X) \geq E_{X' \in succ(X, besta(X))}(c(S(X), besta(X), S(X')) + v(X'))$ otherwise.*

Theorem 3 *v-values of states are monotonically non-decreasing but are bounded from above by the corresponding v^u-values.*

After each iteration of the algorithm the value of state X_{p} (and possibly others) is corrected by either changing its $besta$ pointer or making an increase in its v-value. The number of possible actions is finite. The increases, on the other hand, are bounded from below by a positive constant because the belief state-space is finite (since we assumed perfect sensing and a finite number of possible values for each hidden variable). Therefore, the algorithm terminates. Moreover, at the time of termination v-value of every state on the policy is no smaller than the expectation over the immediate cost plus v-value of the successors. Therefore, the expected cost of the policy can not be larger than $v(X_{\mathrm{start}})$. This is summarized in the following theorem.

Theorem 4 *PPCP terminates and at that time the cost of the policy defined by $besta$ pointers is bounded from above by $v(X_{\mathrm{start}})$ which in turn is no larger than $v^u(X_{\mathrm{start}})$.*

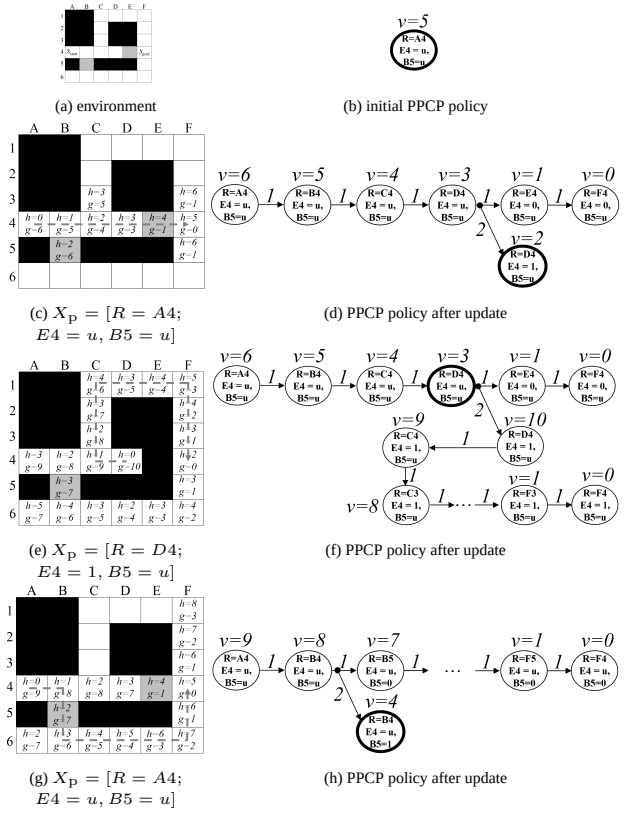


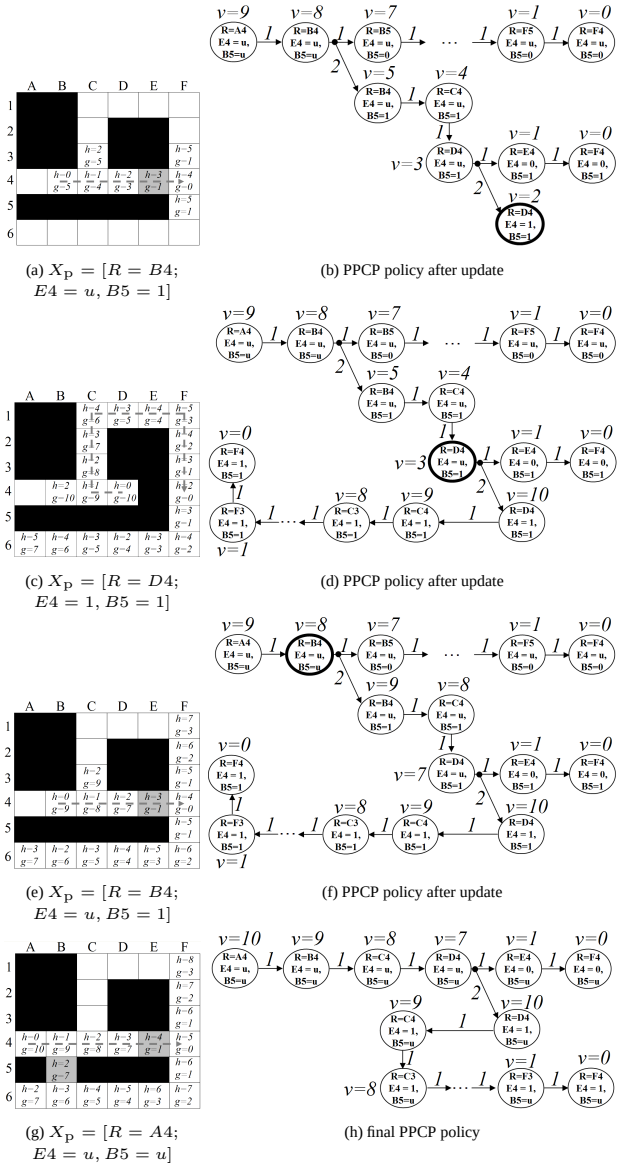
Figure 4: An example of PPCP operation

The final theorem states that the found policy is optimal if the memory about preferred outcomes is not required in an optimal policy. We use $\pi(X)$ to denote a pointer to the action dictated by policy π at the belief state X.

Theorem 5 *Suppose there exists a minimum expected cost policy π^* that satisfies the following condition: for every pair of states $X_1 \in \pi^*$ and $X_2 \in \pi^*$ such that X_2 can be reached with a non-zero probability from X_1 when following policy π^* it holds that the hidden variable $h^{S(X_1), \pi^*(X_1)}$ is not the same as $h^{S(X_2), \pi^*(X_2)}$ or both actions are independent of the values of the hidden variables. Then the policy defined by $besta$ pointers at the time PPCP terminates is also a minimum expected cost policy.*

Example

Figures 4 and 5 demonstrate the operation of the presented PPCP algorithm on a simple instance of the robot navigation in partially-known terrain problem. The robot is in the cell $A4$ and its goal is the cell $F4$. There are two cells (shaded in grey) whose status is unknown to the robot: cell $B5$ and $E4$. For each, the probability of containing an obstacle is 0.5. In this example, we restrict the robot to move only in four compass directions. Whenever the robot attempts to enter an unknown cell, we assume the robot moves towards the cell, senses it and enters it if it is free and returns back otherwise. The cost of each move is 1, the cost of moving towards an unknown cell, sensing it and then returning back is 2. With this setup of costs, a clear preference is for each unknown cell to be traversable. These preferences satisfy def.



(a) $X_{\rm p} = [R = B4; E4 = u, B5 = 1]$ (b) PPCP policy after update

(c) $X_{\rm p} = [R = D4; E4 = 1, B5 = 1]$ (d) PPCP policy after update

(e) $X_{\rm p} = [R = B4; E4 = u, B5 = 1]$ (f) PPCP policy after update

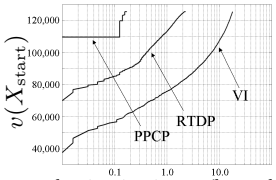
(g) $X_{\rm p} = [R = A4; E4 = u, B5 = u]$ (h) final PPCP policy

Figure 5: An example of PPCP operation (cont'd)

1 required by PPCP algorithm. In the left column of each figure we show the environment each ComputePath function sets up when executed on $X_{\rm p}$ (shown underneath the figure). We also show the heuristics (shown as h-values), the g-values and the path from $S(X_{\rm p})$ to $S_{\rm goal}$ (shown in grey dotted line) computed by the search. In the right column of each figure we show the current policy found by PPCP after the UpdateMDP function incorporates the results of the most recent search (the search shown in the left column in the same row). The states whose v-values are smaller than what they are supposed to be according to their successors are outlined in bold. These states are candidates for being $X_{\rm pivot}$ in the next search iterations. We exercise the following simple optimization in this example and all experiments: whenever a state X is chosen as the next $X_{\rm pivot}$, we backtrack from X up along the policy until the first stochastic transition (or $X_{\rm start}$, whichever comes first), at which point

# of unknowns	Percent Solved			Time to Convergence (in secs)			Solution Cost
	VI	RTDP	PPCP	VI	RTDP	PPCP	Same for All
6	92%	100%	100%	7.7	0.4	0.1	112,284
10	—	92%	100%	—	19.7	0.2	117,221
14	—	80%	100%	—	25.8	0.2	113,918
18	—	48%	100%	—	52.3	0.7	112,884

(a) runs on small environments



(b) rate of convergence

# of unknowns	Traversal Cost	
	PPCP	Freespace
1,000 (0.4%)	1,368,388	1,394,455
2,500 (1.0%)	1,824,853	1,865,935
5,000 (2.0%)	1,521,572	1,616,697
10,000 (4.0%)	1,626,413	1,685,717
25,000 (10.0%)	1,393,694	1,484,018

(c) runs on large environments

Figure 6: Experimental Results

$X_{\rm pivot}$ is chosen to be the outcome of this transition that resides on the same branch as X. For example, in figure 5(d) $X_{\rm pivot}$ is chosen to be state $[R = B4; E4 = u, B5 = 1]$ (robot is at $B4$, cell $E4$ is unknown and cell $B5$ is known to contain an obstacle) as a result of backtracking from state $[R = D4; E4 = u, B5 = 1]$. In computing $\tilde{Q}_{v,g}$, the v-values of belief states that do not yet exist are assumed to be equal to the Manhattan distances to the goal. These distances are also used to initialize the v-values of new belief states. Figure 5(h) shows the policy that PPCP returns after it converges. While it is optimal in this case, it is possible to set up an environment when it will be sub-optimal. An optimal policy for such case would require the robot to sense a cell but then come back from it, and use the fact that the cell is free at a later time. Another example, where the robot needs to remember the status of sensed cells is when it has a long range sensor. A simple solution for this scenario is to incorporate into $S(\cdot)$ the information about the last few sensed cells. While it certainly increases the size of the state-space that each search needs to handle, based on our recent experiments PPCP can still scale up to very large environments with large number of unknowns.

Experimental Study

We have used the problem of robot navigation in unknown terrain to evaluate the performance of our algorithm. In all of the experiments we used randomly generated fractal environments that are often used to model outdoor environments (Stentz 1996). In the first set of experiments we compared the performance of our algorithm with two optimal algorithms: VI (value iteration) with a simple reachability analysis and RTDP (Barto, Bradtke, & Singh 1995), both planning in the belief state-space. Figure 6(a) shows the time it takes to converge, the percent of solved environments (the environments were declared to be unsolved when an algorithm ran out of memory) and the solution cost for the three algorithms for the environments of size 17 by 17 cells. The number of unknown locations increases from 6 to 18 and for each number the results are averaged over 25 environments. The figure shows that PPCP converges faster than the other

algorithms and the differences in speeds grow large pretty fast with the increase in the number of unknown locations. More importantly, PPCP was able to solve all environments in all cases. (We don't give numbers for VI for more than 6 unknowns because it was running out of memory on almost all environments.) Figure 6(a) also shows that in all the cases the solution returned by PPCP turned out to be the same as the one returned by RTDP and VI, an optimal solution. Finally, Figure 6(b) shows the rate of convergence (v-value of start state) of the algorithms for one of the environments with 6 unknowns (note the log scale of the time). Besides RTDP there are other efficient algorithms such as LAO* (Hansen & Zilberstein 2001), HDP (Bonet & Geffner 2003) and MCP (Likhachev, Gordon, & Thrun 2004) that can be used to plan in finite belief state-spaces. While we have not compared their performance we believe they would show the performance similar to the one exhibited by RTDP since they all *have* to perform planning in the belief state-spaces that are exponential in the number of unknowns.

The second set of experiments shows that PPCP can be applied to the problem of robot navigation in environments of large size and with large number of unknown locations. Figure 6(c) compares the performance of PPCP against a strategy of planning with freespace assumption. The comparison is done on the environments of size 500 by 500 cells with the number of unknown locations ranging from 1,000 (0.4% of overall size) to 25,000 (10%). Unlike in the previous experiments, in these ones the robot was moving and was given only 1 second to plan during each of its moves. This amount of time was always sufficient for planning with freespace assumption to generate a path. The PPCP planning, however, was interleaved with execution in the following way. The robot executed PPCP for 1 second. After 1 second, the loop in the Main function of PPCP was suspended (right after the UpdateMDP function returns) and the robot started following the policy currently found by PPCP. During each robot move, the X_{start} state maintained by PPCP was updated to the robot's current state and the main loop of PPCP was resumed for 1 second. After it was suspended again, the policy that the robot follows was updated based on the results of PPCP planning. After a number of robot moves, PPCP algorithm usually converged to a final policy, at which point PPCP was no longer executed by the robot. Figure 6(c) summarizes the execution costs of two approaches averaged over 25 randomly generated fractal environments for each row in the table. The results show that the cost of the trajectory traversed by the robot with PPCP planning is consistently smaller than the one traversed by the robot with freespace assumption planning.

Related Work

In general, planning under uncertainty about the environment corresponds to the problem of planning for partially observable Markov Decision Processes (POMDPs). Planning optimally for POMDPs is known to be hard (PSPACE-complete (Papadimitriou & Tsitsiklis 1987)), and various approximations techniques have been recently proposed instead (Roy & Gordon 2002; Pineau, Gordon, & Thrun 2003; Spaan & Vlassis 2004). The problem we are addressing in this paper is a narrower one as we assume that the underlying problem is deterministic and there is only uncertainty about some actions due to missing information about the environment. We also assume sensing is perfect which entails a finite size belief state-space. This type of problem was formalized well in (Ferguson, Stentz, & Thrun 2004) in the framework of robot navigation in unknown terrain. In fact, their work is most relevant to ours since their planner has also taken advantage of the idea that the value of the plan with a free unknown cell can not be larger than the value of the plan without it. They proposed a clever planner that is capable of finding optimal policies several orders of magnitude faster than other optimal approaches. The goal of the present work, however, is to avoid dealing with the exponentially large belief state-spaces altogether, which is required to guarantee the optimality of the solution. Our aim was to develop an anytime algorithm that can handle very large environments with the substantial amount of uncertainty at the expense of the solution optimality guarantee.

References

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bonet, B., and Geffner, H. 2003. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, 1233–1238.

Ferguson, D.; Stentz, A.; and Thrun, S. 2004. PAO* for planning with hidden state. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*.

Hansen, E., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129:35–62.

Koenig, S., and Smirnov, Y. 1996. Sensor-based planning with the freespace assumption. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Likhachev, M., and Stentz, A. 2003. PPCP algorithm with formal analysis. Tech. Rep., Carnegie Mellon University, Pittsburgh, PA.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. Planning for markov decision processes with sparse stochasticity. In *Advances in Neural Information Processing Systems (NIPS) 17*. Cambridge, MA: MIT Press.

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of Markov decision processses. *Mathematics of Operations Research* 12(3):441–450.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: An anytime algorithm for pomdps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Roy, N., and Gordon, G. 2002. Exponential family pca for belief compression in pomdps. In *Advances in Neural Information Processing Systems*.

Spaan, M., and Vlassis, N. 2004. A point-based POMDP algorithm for robot planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2399–2404.

Stentz, A. 1996. Map-based strategies for robot navigation in unknown environments. In *AAAI Spring Symposium on Planning with Incomplete Information for Robot Problems*.