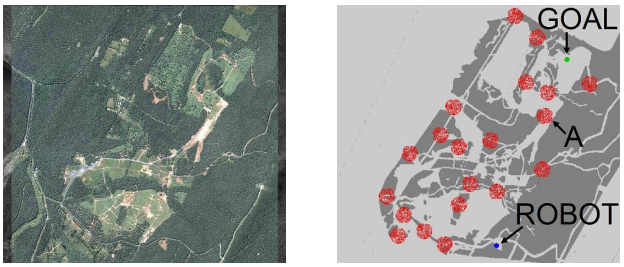
Goal Directed Navigation with Uncertainty in Adversary Locations

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

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

***Abstract*— This paper addresses the problem of planning for goal directed navigation in the environment that contains a number of possible adversary locations. It first shows that commonly used approaches such as assumptive planning can result in very long and costly robot traverses. It then shows how one can solve the same problem using a general probabilistic planner we have recently developed called PPCP (Probabilistic Planning with Clear Preferences). The paper also introduces two optimizations to the PPCP algorithm that make it run up to five times faster for our domain. The experimental results show that solving the problem with PPCP can substantially reduce the expected execution cost as compared to assumptive planning.**



(a) 3.5 by 3.0 km satellite image (b) corresponding traversability map

Fig. 1. Path clearance problem

I. MOTIVATION

This paper addresses the problem of planning for a robot whose task is to reach its goal as quickly as possible without being detected by an adversary. The robot does not know beforehand the precise locations of adversaries, but has a list of their possible locations. When navigating, the robot can come to a possible adversary location, sense it using its long range sensor and go around the area if an adversary is detected or cut through this area otherwise. We will refer to this problem as *path clearance*.

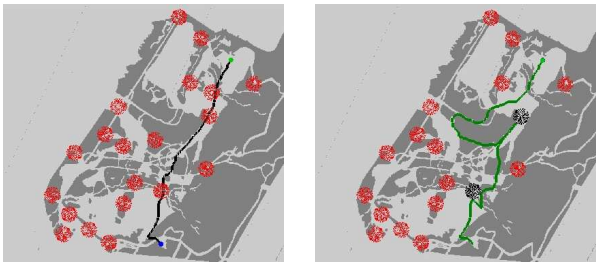
The example in figure 1 demonstrates the path clearance problem. Figure 1(b) shows the traversability map of the satellite image of a 3.5km by 3km area shown in figure 1(a). The traversability map is obtained by converting the image into a discretized 2D map where each cell is of size 5 by 5 meters and can either be traversable (shown in light grey color) or not (shown in dark grey color). The robot is shown by the blue circle and its goal by the green circle. Red circles are *possible* adversary locations and their radii represent the sensor range of adversaries (100 meters in this example). The radii can vary from one location to another. The locations can be specified either manually or automatically in places such as narrow passages. Each location also comes with a probability of containing an adversary (50% for each location in the example): the likelihood that the location contains an adversary. The probabilities can vary from one location to another.

The path the robot follows may change any time the robot senses a possible adversary locations (the sensor range of the robot is 105 meters in our example). A planner, therefore, needs to reason about possible outcomes of sensing and generate a plan (policy) that dictates which path the robot should take as a function of the outcome of each sensing. Ideally, the generated policy should minimize the expected traversal distance. Unfortunately, such planning problem falls into a broad class of POMDP (Partially Observable Markov Decision Processes) problems which are intractable to solve [9]. In fact, the path clearance problem is very much equivalent to the problem of planning for a robot navigating in a partially-known (or unknown) environment: the robot needs to reach its goal but it is initially uncertain about the traversability of some (or all) of the areas of the environment. The difference is that in the path clearance problem, detecting an adversary blocks a large area resulting in a long detour. An adversary location has also a tendency to be placed in such places that it blocks the whole path and the robot has to backup and choose a totally different route. As a result, the detours can be much costlier than in the case of navigation in a partially-known environment, even when the amount of uncertainty is much less.

II. USING ASSUMPTIVE PLANNING

To avoid the computational complexity, a robot operating in a partially-known environment often performs assumptive planning [8], [4], [10]. In particular, it often just follows a shortest path under the assumption that all unknown areas in the environment are free unless the robot has already sensed them otherwise. This is known as a freespace assumption [4]. The robot follows such path until it either reaches its goal or senses new information about the environment. In the latter case, the robot re-computes and starts following a new shortest path under the freespace assumption.

The freespace assumption is also applicable to the path clearance problem. The robot can always plan a path under the assumption that no adversary is present unless sensed otherwise. This assumption makes path clearance a deterministic planning problem and therefore can be solved efficiently. The fact that the robot ignores the uncertainty about the



(a) planned path (b) actual robot's path

Fig. 2. Solving path clearance problem with freespace assumption

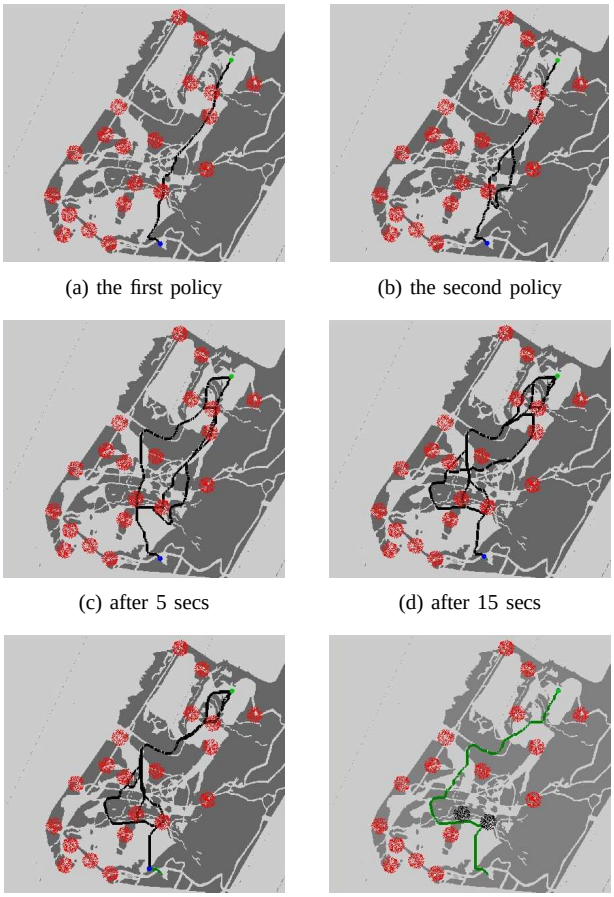
adversaries, however, means that it risks having to take long detours, and the detours in the path clearance problem tend to be longer than in the problem of navigation in a partially-known environment as we have previously explained.

For example, figure 2(a) shows the path computed by the robot that uses the freespace assumption. According to the path, the robot tries to go through the possible adversary location A (shown in figure 1(b)) as it is on the shortest route to the goal. As the robot senses the location A, however, it discovers that the adversary is present in there (the red circle becomes black after sensing). As a result, the robot has to take a very long detour. Figure 2(b) shows the actual path traversed by the robot before it reaches its goal.

III. USING PPCP PLANNING

While the general decision-theoretic planning that takes into account the uncertainty about the environment corresponds to POMDP planning and is therefore hard to solve, it turns out that many such problems exhibit a special property: one can clearly identify beforehand the best (called *clearly preferred*) values for the variables that represent the unknowns in the environment. For example, in the problem of navigation in partially-known environments, it is always preferred to find out that an initially unknown location is traversable rather than not. In a very similar problem of robot navigation in office-like environments with uncertainty about whether some doors are open or not, it is always preferred to find out that a door is open. The same property holds for the path clearance problem: there are also clear preferences for the values of unknowns. The unknowns are m binary variables, one for each of the m possible adversary locations. The preference for each of these variables is to have a value false: no adversary is present. PPCP (Probabilistic Planning with Clear Preferences) [5] is a recently developed algorithm that scales up to large problems with a significant amount of uncertainty by exploiting this property.

PPCP constructs and refines until converges a policy by running a series of A*-like 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 policy to be valid, it is guaranteed to be optimal if the assumption holds. In the problem of robot navigation in a partially-known environment, PPCP was also shown to nearly always return an optimal policy in the environments small enough to be solved with methods guaranteed to converge to an optimal solution (such as RTDP (Real-Time Dynamic Programming) [1]) [5].



(a) the first policy (b) the second policy

(c) after 5 secs (d) after 15 secs

(e) after 30 secs (PPCP converged) (f) actual robot's path

Fig. 3. Solving path clearance problem with PPCP

Figure 3 shows the application of PPCP to the path clearance example in figure 1. Before the robot starts executing any policy, PPCP plans for five seconds. Figure 3(a) shows the very first policy produced by PPCP (in black color). It is a single path to the goal, which in fact is exactly the same as the path planned by planning with the freespace assumption (figure 2(a)). PPCP produced this path within few milliseconds by executing a single A*-like deterministic search. At the next step, PPCP refines the policy by executing a new search which determines the cost of the detour the robot has to take if the first adversary location on the found path contains an adversary. The result is the new policy (figure 3(b)). PPCP continues in this manner and at the end of five seconds allocated for planning, it generates the policy shown in figure 3(c). This is the policy that is passed to the robot for execution. Each fork in the policy is where the robot tries to sense an adversary and chooses the corresponding branch.

As we will explain in the later section, we interleave planning with execution. Thus, while the robot executes the plan, PPCP improves it relative to the current position of the robot. Figure 3(d) shows the new position of the robot (the robot travels at the speed of 1 meter per second) and the current policy generated by PPCP after 15 seconds since the robot was given its goal. Figure 3(e) shows the position of the robot and the policy PPCP has generated after 30 seconds. At this point, PPCP has converged and no more refinement is necessary. Note how the generated policy makes the robot go through the area on its left since there are a number of ways to get to the goal and therefore there is a high chance that one of them will be available. Unlike the plan generated by planning under freespace assumption, the plan generated by PPCP avoids going through location A. Figure 3(f) shows the actual path traversed by the robot. It is 4,123 meters long while the length of the trajectory traversed by the robot that plans with freespace assumption (figure 2(b)) is 4,922 meters.

IV. PPCP ALGORITHM

In this section we will describe the previously developed PPCP algorithm [5]. To better illustrate the algorithm, we will use the simple example of the robot navigation problem in a partially-known environment in figure 4(a). The robot is in cell A4 and its goal is 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. (In the path clearance problem, we allowed the robot to move in eight directions. Our approach of using PPCP is also equally applicable when the robot transitions and the discretization of the environment are more complex than those in a gridworld.) Whenever the robot attempts to enter an unknown cell, we assume that 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.

1) Assumptions and Notations: PPCP assumes that the environment itself is fully deterministic and can be modeled as a graph. There are certain elements of the environment, however, whose status we are uncertain about and which affect the outcomes and possibly costs of one or more actions. Thus, in the figure 4 example, each of the robot transitions is deterministic if it is known whether the target cell is free or not. If, however, the robot tries to enter the cell whose status it does not know about, then it may or may not be able to enter it, depending on its actual status. Thus, the action of moving towards such cell needs to be treated by the planner as stochastic. These assumptions are clearly a relaxation of the assumptions made when planning with the freespace assumption, which assumes that the robot always moves deterministically and all unknown cells are free. (Both, PPCP and planning with the freespace assumption, have to re-plan whenever the robot deviates from its plan due to actuation errors.)

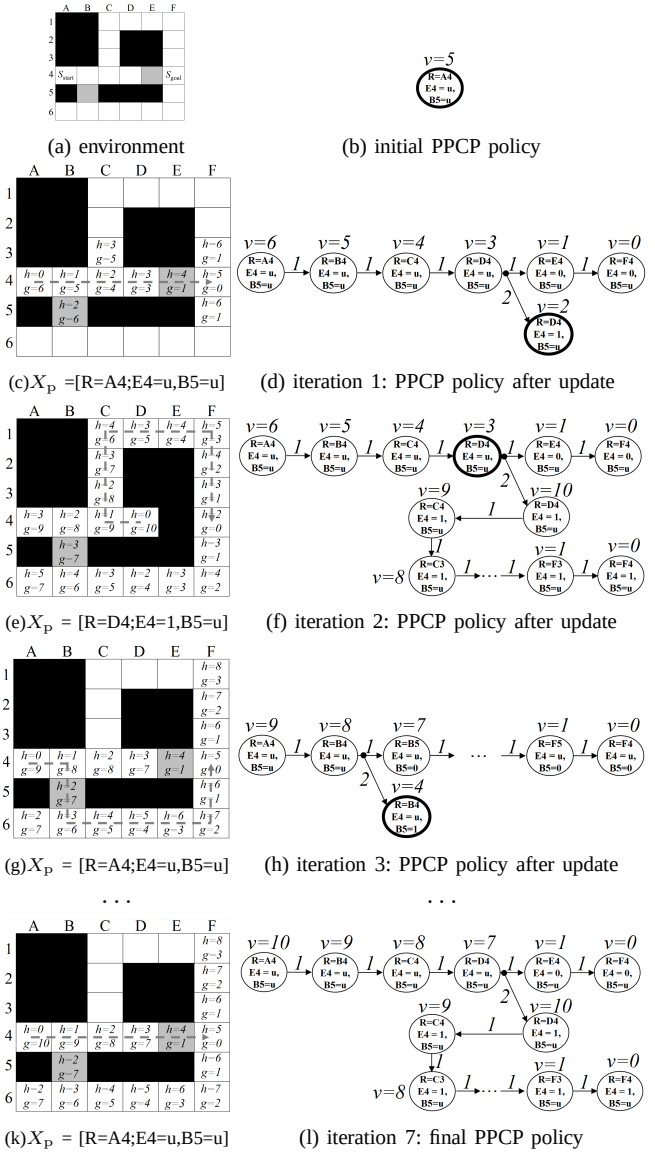


Fig. 4. An example of PPCP operation. Figures (c,e,g,k) show the states computed by the ComputePath function in each iteration. The right column shows the policies generated by PPCP after each iteration.

In the notations used by PPCP X is a full state-vector (a belief state). It can be split into two sets of (discrete and of finite-range) 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 represent the missing information about the environment. In the figure 4 example, these correspond to the status of unknown cells B5 and E4. $h_i(X)$ denotes an i^{th} variable in $H(X)$. We will use the setting $h_i(X) = u$ to 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. Thus, in the figure 4 example, each unknown cell is represented by its own hidden variable. Whenever the robot senses a previously unobserved cell, the value of the corresponding variable h_i transitions from u to either 0 or 1 and remains there afterwards. (If at a later time, the status of the cell

changes, PPCP will have to re-plan.)

X_{start} is used to denote the start state: in the figure 4 example, 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 figure 4, S_{goal} corresponds to the robot being at cell $F4$.

The actions are associated with $S(X)$ rather than the full belief state X since the values of variables in $H(X)$ affect purely the costs and outcomes of actions. We use $succ(X, a)$ to denote the set of outcomes (successors) of action a taken in a belief state X. If the execution of action a does not depend on any of the variables h_i whose values are not yet known at X, then $succ(X, a)$ contains only one state Y and $H(Y)$ is the same as $H(X)$. Otherwise, if the outcome of action a depends on the value of a variable h_i whose value is not yet known at X, then $succ(X, a)$ may contain two or more states. The value of h_i becomes known at these states. The probability distribution of these outcomes follows the probability distribution of h_i. Thus, in figure 4, the action a of moving east from cell A4 has only one outcome since it does not depend on the values of any of the hidden variables. On the other hand, the action a of moving east at state $X = [R = D4; B5 = u, E4 = u]$ (that is, robot at cell D4 and the status of both initially unknown cells is still unknown) has two equally probable outcomes - $Y_1 = [R = E4; B5 = u, E4 = 0]$ and $Y_2 = [R = D4; B5 = u, E4 = 1]$ - because the probability of cell $E4$ being blocked is 0.5. PPCP assumes that the variables in H can be considered independent of each other. In other words, the status of cell B5 is independent of the status of cell E4. Costs need to be associated with $S(X)$. Thus, $c(S(X), a, S(Y))$ denotes the cost of executing action a at any belief state whose observable state is equal to $S(X)$ and moving into a belief state whose observable state is equal to $S(Y)$.

The main assumption that PPCP makes is that clear preferences on the values of the hidden variables are available. It requires that for each variable $h_i \in H$ we are given its preferred value, denoted by b (i.e., the best value). This value must satisfy the following property. Given any state X and any action a whose outcomes depend on the value of h_j and its value is not yet known at X (i.e., $h_j(X) = u$), there exists a successor state X' such that $h_j(X') = b$ 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. PPCP uses the notation $succ(X, a)^b$ (i.e., the best successor) to denote such state X'. If the value of h_j is known at X (i.e., $h_j(X) \neq u$), then $succ(X, a)^b$ refers to the only state contained in $succ(X, a)$, independently of the value of its h_j. The robot navigation problem in figure 4 clearly satisfies the property of clear preferences since for each sensing action there always exist two outcomes: a cell is blocked or unblocked. The latter is clearly the preferred outcome.

```
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 the priority 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 a and S(Y) s.t. S(X) = S(succ(Y^u, a)^b)
8      compute Q̃_{v,g}(S(Y), a) according to equation 1;
9      if this search hasn't seen S(Y) yet or g(S(Y)) > Q̃_{v,g}(S(Y), a)
10       g(S(Y)) = Q̃_{v,g}(S(Y), a); besta(S(Y)) = a;
11       insert/update S(Y) in OPEN with the priority
                                        g(S(Y)) + heur(S(X_p), S(Y));
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_{Y∈succ(X,besta(X))} c(S(X), besta(X), S(Y)) + v(Y);
23   if found set X_pivot to X;
24   otherwise set X_pivot to null;
```

Fig. 5. The PPCP Algorithm

2) The algorithm: The pseudocode for the PPCP algorithm is given in figure 5 and a trace of the algorithm operation is given in figure 4. The Main function of the algorithm constructs and refines a policy in the full belief state-space. For each state it maintains a v-value, which is an estimate of the expected cost-to-goal, and $besta$ pointer, a pointer to the action that should be executed at the state according to the current policy. 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). Thus, in figure 4(b) the initial policy is just state X_{start} whose $besta(X_{\text{start}}) = \textbf{null}$. Its initial v-value (and the initial v-values of all other states as they get created) is set to the Manhattan distance from the state to the goal.

The Main function of the algorithm repeatedly executes A*-like searches by calling the ComputePath function 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 22). If a state has $besta = \textbf{null}$ then the value of expectation over policy action successors is assumed to return ∞. Thus, X_{start} is the first state to be selected for calling the ComputePath function on (figure 4(c)).

The ComputePath function is nearly identical to (backward) A* search. While the function is called on a belief state X_{p}, it searches backwards from a single state, S_{goal}, towards a single state $S(X_{\text{p}})$, and the states in the search state-space consist only of variables in S. Thus, in the robot navigation example it means that the search is actually a 2D search, and not a search in a much higher dimensional belief state-space. Consequently, the search is fast. The search is also deterministic because it assumes that any action a executed at $S(X)$ has only one outcome. If the value of a hidden variable h_j that affects the outcomes of a was known at state

$X_{\rm p}$, then the search uses the outcome that corresponds to the value of $h_j(X_{\rm p})$. Otherwise, the search makes an optimistic assumption and uses the preferred outcome. One way to put it formally is to say that the search assumes that the outcome of action a executed at state $S(X)$ is $S(succ(X^u, a)^b)$, where X^u is a state whose $S(X^u) = S(X)$ and $H(X^u)$ is $H(X_{\rm p})$ but with each hidden variable equal to b set to u. Thus, when executed on $X_{\rm p} = [R = D4; E4 = 1, B5 = u]$ in figure 4(e), the ComputePath function assumes cell E4 is blocked whereas cell B5 is free. After the environment is set, the ComputePath function performs exactly backward A* search in this environment with the only exception of how the g-values are computed.

In the backward A* search, g-values are estimates of the cost-to-goal distances. Thus, if we were to run a normal backward A* search on the environment the ComputePath function sets up, then the g-value of state $S(Y)$ with a successor state $S(X) = S(succ(Y^u, a)^b)$ according to the search tree generated by the search is given by

$$g(S(Y)) = c(S(Y), a, S(X)) + g(S(X))$$

We'll denote the right hand side of this equation as $\tilde{Q}_{v,g}(S(Y), a)$. The ComputePath function, on the other hand, computes the g-value of state $S(Y)$ differently, namely as:

$$\tilde{Q}_{v,g}(S(Y), a) = \sum_{Z \in succ(Y^u, a)} P_{Y^u, a}(Z) \cdot \max(c(S(Y), a, S(Z)) + v(Z), \; c(S(Y), a, S(Z^b)) + g(S(Z^b))) \quad (1)$$

where $Z^b = succ(Y^u, a)^b$. In case when action a executed at Y^u is deterministic, this equation reduces exactly to the equation used in normal A* search. Otherwise, this equation incorporates the values of other outcomes of action a besides the one that is preferred. For example, in figure 4(g) the g-value of cell C4 is 8, which is $1+g(D4)$. The g-value of cell D4, on the other hand, is 7, despite the fact that $g(E4) = 1$, and the reason is that the v-value of the state $[R = D4; E4 = 1, B5 = u]$ that corresponds to the "bad" outcome of going east at cell D4 is 10. The v-value of this state has just been updated in the previous iteration (figure 4(f)).

Apart from the computation of g-values, the ComputePath function is fully equivalent to the A* search. It also uses the heuristics that estimate distance to $S(X_{\rm p})$. The heuristics need to satisfy normal consistency requirements [7]. In figure 4 they are shown as h-values and are Manhattan distances. In the pseudocode they are referred to as $heur$ variables.

After each search, the UpdateMDP function updates the v-values of the states that belong to the path found by the search by setting them to their g-values. This increases v-values and is guaranteed to correct the error between the v-values of the states on the path and the v-values of their successors in the policy. The iterations continue until there are no states left on the current policy whose values are smaller than what they should be according to their successors. The algorithm is guaranteed to terminate (the proof of this and other properties of the algorithm can be found in [6]). At the time of termination, the expected cost of the found policy is bounded from above by the cost of the policy in which the robot always forgets the outcome of sensing if it was a preferred outcome. If an optimal policy does not require remembering preferred outcomes, then the policy returned by PPCP is also guaranteed to be optimal. In the figure 4 example, this memoryless property would mean that during the planning process the planner assumes that as soon as the robot successfully enters cell E4 and therefore finds out that is free, it resets back the status of cell E4 to an unknown cell. (The non-preferred outcomes of sensing are never reset and so if the robot finds that the cell E4 is blocked, then this information is retained forever.) In the example, an optimal plan does not need to remember the status of any cell the robot has successfully entered. In the path clearance problem, however, this is not the case and we need to address the memoryless property of PPCP.

V. APPLICATION OF PPCP TO PATH CLEARANCE

A. Overcoming memoryless property

PPCP assumes that the robot does not need to remember the outcomes of sensing if they are preferred outcomes. In the figure 4 example, for instance, the robot senses a cell when trying to enter it. If it is free then the robot enters it and does not really need to remember that it is free afterwards, its projected path is unlikely to involve entering the same cell again. It only needs to remember if some cells turn out to be blocked, and plans generated by PPCP do retain this information.

This is different, however, when the robot has a long range sensor such as in the path clearance problem. The robot may sense whether an adversary is present before actually reaching the area covered by the adversary. It thus needs to remember the preferred outcomes if it wants to sense adversaries from a long distance.

The solution to this problem is to augment each state $S(X)$ with the last k preferred outcomes of sensing. Thus, $S(X)$ now consists of the location of the robot plus the last k locations that were sensed to be free of adversaries. k can be set to a small number such as two or three to keep the complexity of each search low. This amount of memory about the preferred outcomes of sensing is enough for the path clearance problem.

B. Interleaving planning and execution

In many cases we would like to be able to interleave planning with execution. The robot can then start executing whatever current plan it has and while executing it, a planner can work on improving the plan. This way the robot does not need to wait until the planner fully converges.

Interleaving planning with PPCP with execution can be done as follows. The robot first executes PPCP for several seconds. The loop in the Main function of PPCP is then suspended (right after the UpdateMDP function returns on line 21), and the robot starts following the policy currently found by PPCP as given by $besta$ pointers. During each robot move, $X_{\rm start}$ state maintained by PPCP is updated to

the current state of the robot and the main loop of PPCP is resumed for another few seconds. After it is suspended again, the policy that the robot currently follows is compared against the policy that PPCP currently has and is updated to it only if the latter has a higher probability of reaching a goal location. If the policy the robot currently follows has a higher probability of reaching the goal then the robot continues to follow it. This way we avoid changing policies every time PPCP decides to explore a different policy but has not explored much of the outcomes on it yet. The probabilities of reaching a goal for an acyclic policy can be computed in a single pass over the states on the policy in their topological order.

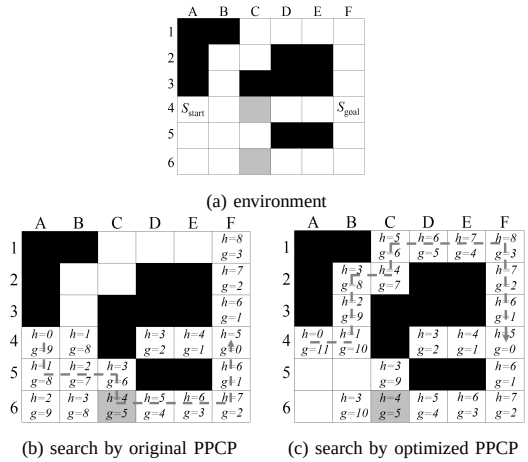
Once PPCP converges to a final policy, the robot can follow the policy without re-executing PPCP again unless the robot deviates from its path due to actuation errors. If the robot does deviate significantly from the plan generated by PPCP, then PPCP can be used to re-plan. There is no need to re-plan from scratch. Instead, X_{start} is updated to the new state of the robot, the old policy is discarded, and the main loop of PPCP is resumed. Note that the values of all state variables in PPCP are preserved. It will then automatically re-use them to find a new policy from the current robot position much faster than if PPCP was re-executed from scratch.

C. Optimizations

We now describe two general optimizations of the PPCP algorithm that prove to be very effective for the path clearance problem. The first one reduces the number of search iterations PPCP makes, while the second optimization substantially speeds up each search. Both optimizations leave the theoretical properties of the algorithm such as convergence and optimality under certain conditions unchanged and can be considered general optimizations of PPCP.

1) Reducing the Number of Search Iterations: As mentioned previously, during each search whenever the ComputePath function encounters action a executed at state $S(X)$ and the outcome is not definite according to $H(X_{\text{p}})$, then in evaluating the equation 1, the ComputePath function uses the v-values of non-preferred outcomes. The v-values are estimates of the goal distances. If these non-preferred outcomes have never been explored by PPCP, then the v-values are initial estimates of the cost-to-goal from them and are likely to be much lower than what they should really be. This means that the ComputePath function will return a path that uses the state-action pair $(S(X), a)$ and only in the future iterations PPCP will find out that the v-value of these non-preferred outcomes should really be higher and this state-action pair should be avoided.

Figure 6 gives such example for the robot navigation in a partially-known environment problem. When solving the environment in figure 6(a), PPCP at some point invokes the ComputePath function on state $X_{\text{p}} = [R = A4; C4 = 1, C6 = u]$. Suppose that by this time PPCP has already computed the v-value of state $[R = B6; C4 = 1, C6 = 1]$ as 12. This is the cost of getting to the goal from cell B6 if both cells C4 and C6 are blocked. During the current search,



(a) environment

(b) search by original PPCP (c) search by optimized PPCP

Fig. 6. The comparison of a search by the original PPCP (b) and PPCP with the optimization described in section V-C.1

when computing the g-value of cell C5, PPCP will query the v-value of state $[R = C5; C4 = 1, C6 = 1]$ as needed by the equation 1. If the v-value of this state has never been computed previously, PPCP initializes it to some admissible estimate such as Manhattan distance from C5 to the goal cell, which is 4 (figure 6(b)). After evaluating equation 1, the g-value of cell C5 becomes 6 ($= 0.5 \max(2 + 4, 1 + 5) + 0.5 \max(1+5, 1+5)$). Consequently, the search returns the path shown in figure 6(b) that goes through cells C5 and C6.

The optimization that we propose in this section is to use the v-values of neighboring states to obtain more informative v-values of states that have not been explored yet. Thus, in the example, we can deduce that the v-value of state $[R = C5; C4 = 1, C6 = 1]$ can be at least 10 - the v-value of state $[R = B6; C4 = 1, C6 = 1]$, which is 12, minus an upper bound on the cost of getting from $[R = B6; C4 = 1, C6 = 1]$ to state $[R = C5; C4 = 1, C6 = 1]$, which we can easily compute as 2. More formally, suppose we are interested in estimating the v-value of some state X. We can then take some (small) region R of states around X whose $H(\cdot)$ part is the same as in $H(X)$. Using each state $Y \in R$ and an upper bound $c^u(Y, X)$ on getting from state Y to state X, we can then estimate $v(X)$ as:

$$v(X) = \quad \max_{Y \in R} v(Y) - c^u(Y, X) \tag{2}$$

The upper bounds, $c^u(\cdot, X)$ can be computed via a single backward Depth-First Search from X. In some problems they can also be obtained a priori. To see that the equation 2 is a valid update for $v(X)$ consider the following trivial proof that $v(X)$ remains admissible (does not overestimate the minimum expected cost of getting to goal) provided an admissible value $v(Y)$ for each $Y \in R$. Let $v^*(Y)$ denote the minimum expected cost of getting to the goal from Y. Then:

$$v(Y) \leq v^*(Y) \leq c^u(Y, X) + v^*(X)$$

Thus, $v^*(X)$ is bounded from below by $v(Y) - c^u(Y, X)$ and by setting $v(X)$ to it we guarantee the admissibility of $v(X)$.

The only change to the algorithm is that on line 15 the v-value update is now a maximum between the old v-value and the g-value. In other words, the new line 15 is now as follows:

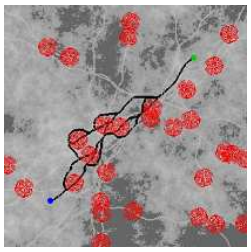
15 $v(X) = \max(v(X), g(S(X)))$; $v(X^u) = \max(v(X^u), g(S(X)))$;
$besta(X) = besta(S(X))$;

Figure 6(c) shows the operation of the ComputePath function that uses our optimization. The g-value of cell C5 now is computed as 9 ($= 0.5 \max(2 + 10, 1 + 5) + 0.5 \max(1 + 5, 1 + 5)$) because it uses the v-value of state $[R = B6; C4 = 1, C6 = 1]$ to better estimate the v-value of $[R = C5; C4 = 1, C6 = 1]$ — the non-preferred outcome of moving from C5 towards C6. Consequently, the search returns a very different path and PPCP never has to explore the path through cells C5 and C6 that would have been returned without our optimization. The proposed optimization can substantially cut down on the overall number of search iterations PPCP has to do. This significantly overcomes the expense of computing better estimates of v-values for non-preferred outcomes.

2) Speeding Up Searches: PPCP repeatedly executes A*-like searches. As a result, much of the search efforts are repeated and it should be beneficial to employ the techniques such as D* [10], D* Lite [2] or Adaptive A* [3] that are known to significantly speed up repeated A* searches. We use the last method because it guarantees not to perform more work than A* search itself and also requires little changes to our ComputePath function.

The idea is simple and is as follows. This optimization computes more informed heuristic values, $heur(S(X))$, that are used to focus each search. $heur(S(X))$ is a heuristic value that (under) estimates a distance from $S(X_{\rm p})$ to $S(X)$ under the assumption that all hidden variables whose values are unknown are set to b. The heuristics need to be consistent [7]. Initially, before any search iteration is done, we compute the start distance (the cost of a least-cost path from $S(X_{\rm start})$ to the state in question) of every state in $S(\cdot)$ assuming that execution of every stochastic action results in a preferred outcome (in other words, the search is done on the deterministic environment where the value of each hidden variable is set to b). In the figure 4 example, it means that we compute the distance from cell A4 to every other cell assuming cells E4 and B5 are free. We can do this computation via a single Dijkstra's search. Let us denote the computed value for each state $S(X)$ by $heur^*(S(X_{\rm start}), S(X))$.

The computed value $heur^*(S(X_{\rm start}), S(X))$ is a perfect estimate of the start distance in the environment where every hidden variable is set to b. Therefore it is a good heuristic value to use when the ComputePath function is invoked with $X_{\rm p} = X_{\rm start}$. The problem, however, is that the ComputePath function is most of the time called to find a path from some other state $X_{\rm p} \neq X_{\rm start}$. We employ the same principle as in [3] that allows us to use our $heur^*$-values anyway: for every state $S(X)$ its heuristic value $heur(S(X))$ can be improved as follows


(a) A typical group I environment

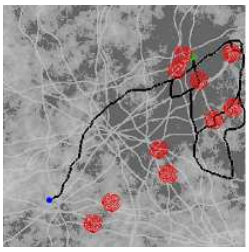

(b) A typical group II environment

Fig. 7. The example of environments used in testing and the plans generated by PPCP for each.

$$heur(S(X)) = \max(heur(S(X)), \\ heur^*(S(X_{\rm start}), S(X)) - heur^*(S(X_{\rm start}), S(X_{\rm p})))$$

(Note that $S(X)$ does not retain the value of $heur(S(X))$ from one search to another.) For the same reasoning as in [3] the updated $heur(S(X))$ is guaranteed not to overestimate the actual distance from $S(X_{\rm p})$ to $S(X)$ and to remain a consistent function.

VI. EXPERIMENTAL RESULTS

In all of our experiments we used randomly generated fractal environments that are often used to model outdoor environments [11]. On top of these fractal environments we superimposed a number of randomly generated paths in between randomly generated pairs of points. The paths were meant to simulate roads through forests and valleys and that are usually present in outdoor terrains. Figures 7(a,b) show typical environments that were used in our experiments. The lighter colors represent more easily traversable areas. All environments were of size 500 by 500 cells, with the size of each cell being 5 by 5 meters.

The test environments were split into two groups. Each group contained 25 environments. For each environment in the group I we set up 30 possible adversary locations at randomly chosen coordinates but in the areas that were traversable. Figure 7(a) shows a plan the PPCP algorithm with our improvements has generated after full convergence for one of the environments in group I. For each environment in the group II we set up 10 possible adversary locations. The coordinates of these locations, however, were chosen such as to maximize the length of detours. This was meant to simulate the fact that an adversary may often be set at a point that would make the robot take a long detour. In other words, an adversary is often set at a place that the robot is likely to traverse. Thus, the environments in group II are more challenging. Figure 7(b) shows a typical environment from the group II together with the plan generated by PPCP. The shown plan has about 95% probability of reaching the goal (in other words, the robot executing the policy has at most 5% chance of encountering an outcome for which the plan had not been generated yet). In contrast to the plan in figure 7(a), the plan for the environment in group II is more complex - the detours are much longer - and it is therefore harder to compute. For each possible adversary location the probability of containing an adversary was set at random to a value in between 0.1 and 0.9.

	# of Expansions	Time to Convergence (secs)	Converged within 15 minutes
PPCP	59,759,717	281.83	64%
optimized PPCP	11,911,585	60.81	92%

TABLE I

THE COMPARISON OF PPCP AND OPTIMIZED PPCP. THE CONVERGENCE TIMES ARE GIVEN FOR THE ENVIRONMENTS ON WHICH *both* ALGORITHMS CONVERGED WITHIN 15 MINUTES.

	Overhead in Execution Cost			
	Group I no penalty	Group II no penalty	Group I with penalty	Group II with penalty
freespace	5.4%	5.2%	35.4%	21.6%
freespace2	0.5%	4.9%	4.8%	17.0%
freespace3	2.1%	4.3%	0.0%	12.7%

TABLE II

THE OVERHEAD IN EXECUTION COST OF NAVIGATING USING PLANNING WITH FREESPACE ASSUMPTION OVER NAVIGATING USING PLANNING WITH PPCP

We have run two sets of experiments on these environments. In the first set we compared the original PPCP algorithm and the PPCP algorithm with the two optimizations we have described in section V-C. Table I shows the results for the group I averaged over all of the environments in it. The algorithms were run until full convergence in order to obtain the comparison results. According to them the number of states expanded by the original PPCP is about five times more and its run-time is also close to five times longer than for the optimized PPCP. The original PPCP has also converged on less environments within 15 minutes.

In the second set of experiments we compared the execution cost of the robot planning with our optimized PPCP versus the execution cost of the robot planning with freespace assumption [4]. In these experiments, unlike in the previous experiments, the robot was moving and was given 5 seconds to plan while traversing 5 meter distance. This amount of time was always sufficient for planning with freespace assumption to generate a path. The PPCP planning, on the other hand, was interleaved with execution as we have explained in section V.

Table II shows the overhead in the execution cost incurred by the robot that plans with the freespace assumption over the execution cost incurred by the robot that uses PPCP for planning. The rows freespace2 and freespace3 correspond to making a cost of going through a cell that belongs to a possible adversary location twice and three times higher than what it really is, respectively. One may scale costs in this way in order to bias the paths generated by the planner with freespace assumption away from going through possible adversary locations. The results are averaged over 8 runs for each of the 25 environments in each group. For each run the true status of each adversary location was generated at random according to the probability having an adversary in there.

The figure shows that planning with PPCP results in considerable execution cost savings. The savings for group I environments were small only if biasing the freespace planner was set to 2. The problem, however, is that the biasing factor is dependent on the actual environment, the way the adversaries are set up and the sensor range of an adversary. Thus, the overhead of planning with freespace for the group II environments is considerable across all bias factors. In the last two columns we have introduced penalty for discovering an adversary. It simulated the fact that the robot runs the risk of being detected by an adversary when it tries to sense it. In these experiments, the overhead of planning with freespace assumption becomes very large. Also, note that the best bias factor for freespace assumption has now shifted to 3 indicating that it does depend on the actual problem. Overall, the results indicate that planning with PPCP can have significant benefits and do not require any tuning.

VII. CONCLUSIONS

In this paper we have shown how to apply the PPCP algorithm to the problem of robot navigation with possible adversary locations and have demonstrated that it is beneficial. The experiments have shown that the algorithm can scale up to large real-world size environments with large number of possible adversary locations, each having long range sensors. We are also currently porting the system onto a real outdoor terrain vehicle. The paper also presents two general optimizations to the PPCP algorithm that can be useful by PPCP when used to solve other planning under uncertainty problems.

VIII. ACKNOWLEDGMENTS

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.

REFERENCES

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

[2] S. Koenig and M. Likhachev. D* lite. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI)*, 2002.

[3] S. Koenig and M. Likhachev. Adaptive A*. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005. Poster abstract.

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

[5] M. Likhachev and A. Stentz. PPCP: Efficient probabilistic planning with clear preferences in partially-known environments. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

[6] M. Likhachev and A. Stentz. PPCP algorithm with formal analysis. Tech. Rep., Carnegie Mellon University, Pittsburgh, PA, 2007.

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

[8] I. Nourbakhsh and M. Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal*, 3(1):49–67, 1996.

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

[10] A. Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

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