Navigation in Partially Observed Dynamic Roadmaps

Bhaskara Marthi
Willow Garage, Inc.
Menlo Park, CA
bhaskara@willowgarage.com

Abstract

We consider robot navigation in environments consisting of a known static map, but where dynamic obstacles of varying and unknown lifespans appear and disappear over time. We describe a roadmap-based formulation of the problem that takes the sensing and transition uncertainty into account, and an efficient online planner for this problem. The planner naturally displays behaviors such as persistence and obstacle timeouts, and is able to make inferences about obstacle types even with impoverished sensors.

1. Introduction

Robot navigation is a well-studied problem, and efficient algorithms exist for known static maps. An autonomous robot in an unconstrained real-world environment such as an office must, however, also deal with various sorts of changes to the map. A person might be standing in a doorway, blocking it. A couch may have temporarily been moved into a corridor, rendering it impassable. And so on. Many such changes can be viewed as adding new obstacles to the world. Navigation algorithms should behave efficiently and sensibly given that such dynamic obstacles appear (and disappear) over time.

The standard approach is to maintain a map, that is updated in some manner given obstacles. Getting this to work robustly is surprisingly tricky, however. Consider a prototypical example. In figure 1, the shortest way to the goal is to go down the narrow hallway. The next best alternative is to go around the building, which takes ten times as long. Now suppose a set of obstacle points appear in the hallway at position A, making it impossible to traverse. Certainly, we don't want the robot to stand and wait forever. The map must therefore be updated with this obstacle, causing the robot to eventually choose the long path. Also, the robot must not suddenly change its mind when it is, say, at position B, and turn back around. This requires either making obstacles fairly long-lived, or hardcoding "persistence" of some sort into the navigation planner. In the first case, there needs to be some scheme for eventually timing out obstacles, to avoid the possibility, e.g., of a corridor being considered permanently out of bounds due to seeing a person blocking it once. Finally, the algorithm should deal

ICAPS 2010 POMDP Practitioners Workshop, May 12, 2010, Toronto, Canada.

Figure 1: An example navigation problem. The goal is to get to G from S.

intelligently with different types of obstacles. It might, for example, make sense to wait for a person to move, but not for a couch. Ideally, perception would give us this information. But even if, as is currently the norm, perception is fairly impoverished/noisy, there is still the possibility of a kind of implicit sensing: in the example, it might make sense to just wait for a few seconds. If the obstacle disappears, we can take the short path after all. If it stays where it is, it is likely to be static and we should take the long route.

Thus, while a reactive approach that ignores the possibility of dynamic obstacles is certainly possible, it requires going beyond simple navigation planning. Essentially, there needs to be a mini-"executive" sitting above the planner, with various hand-coded procedures to avoid infinite loops and dead-ends and to appear intelligent and goal-directed. An alternative is to explicitly model and plan for the uncertainty in the world. This reduces the need for procedurally specified behaviors; instead, the free parameters are part of

a declarative model of the domain that can potentially be learnt from observation. There are many ways to formulate the problem, some of which we discuss in Section 3. One possibility is to model the transition uncertainty with costs or probabilities, while still pretending that the state is fully observable. These approaches are systematically suboptimal, though, as they will not take information-gathering actions. Alternatively, we can explicitly model the sensing uncertainty, using a partially observable Markov decision process (POMDP). POMDPs are notoriously difficult to solve, however, and the POMDP in our case has a state space of size exponential in the number of possible obstacle locations.

In this paper, we present a formulation of the problem as a POMDP, and an algorithm for solving it efficiently enough to run online on a mobile robot. The formulation is based on a topological roadmap over the static map, where nodes represent particular locations and edges are local paths that can become blocked and unblocked over time. Obstacles may belong to different classes, with varying lifetimes. As we will show, the various behaviors discussed above, such as persistence, patience, and implicit sensing, all happen automatically in optimal policies for this POMDP. Our planning algorithm takes advantage of the structure of the problem, specifically that 1) the (topological) position is fully observed 2) observations are spatially local 3) new obstacles appear fairly rarely.

The specific technical contributions are:

- In Section 3, we describe our POMDP formulation of the problem, and show how it leads to reasonable behavior.
- In Section 5, we describe an efficient forward-search planning algorithm for our problem.
- In Section 4, we show how to do efficient state estimation for our model.
- Also in Section 4, we describe a trick for speeding up POMDP forward search algorithms in problems where belief update is expensive.

2. Background

2.1 Continuous Time Markov Chains

We'll model the state of each edge in a roadmap using a (homogeneous) continuous time Markov chain (Norris 1999). A continuous time Markov chain is like a discrete time one, except that the transitions occur at random times, with exponentially distributed gaps. Such a chain is described by an intensity matrix:

$$\begin{pmatrix} -q_1 & q_{12} & \dots & q_{1n} \\ q_{21} & -q_2 & \dots & q_{2n} \\ \dots & \dots & \dots & \dots \\ q_{n1} & q_{n2} & \dots & -q_n \end{pmatrix}$$

Given that we're in state i at time t, the next transition happens at time s where $s - t$ is exponentially distributed with parameter q_i, and the next state is j with probability proportional to q_{ij}. The main fact we will use is that conditional on the history upto time t ending at state i, the probability of being in state j at time $t + s$ is:

$$P(X_{t+s} = j) = (e^{sQ})_{ij} \tag{1}$$

2.2 MDPs

An undiscounted Markov decision process (Puterman 2005), or MDP, consists of a state space S, action set A, transition model $P(s'|s, a)$, reward function $R(s, a, s')$, and terminal states $T \subset S$. A stationary policy is a function $\pi : S \to A$. A policy induces a distribution over state-action trajectories that continue until reaching a terminal state. The value function of a policy $V^\pi(s)$ is the expected total reward for following π starting at s, and the Q-function $Q^\pi(s, a)$ is the expected total reward for doing a in s, then following π. The optimal policy π^* maximizes the value at all states, and we write V and Q for its value and Q-functions. In our examples, the set of actions varies depending on the state, but this can be represented by making nonapplicable actions have reward $-\infty$.

2.3 POMDPs

A partially observable Markov decision process (Kaelbling, Littman, and Cassandra 1998), or POMDP, is like an MDP except that states are not directly observed. Instead, there is an observation distribution $Z(o|s, a, s')$ over the observation that's received when making the transition from s to s' via a.

In the context of POMDPs, we call a distribution over the state space a *belief state*. Given a belief state b about the current state s, if we do an action a, the marginal distribution over the next state is the result of the projection operator $b' = \mathcal{P}(b, a)$ where:

$$b'(s') = \sum_s b(s)P(s'|s, a)$$

Also, given an observation o, the conditional distribution is $b'' = \mathcal{C}(b', s, a, o)$ where:

$$b''(s') = \frac{b'(s')Z(o|s, a, s')}{\sum_{s'} b'(s')Z(o|s, a, s')}$$

The filtering operator $\mathcal{F}$ consists of projection followed by conditioning, and is used to update the distribution over the current state. A key fact about POMDPs is that the sequence of beliefs itself forms an MDP (with the same action space). To sample from the transition model of this MDP given belief b and action a, first sample s from b, then sample s' from $P(\cdot|s, a)$ and o from $Z(\cdot|s, a, s')$, and set $b' = \mathcal{F}(s, a, o)$ (s' is discarded). The reward function is $R(b, a, b') = \sum_{s,s'} b(s)P(s'|s, a)R(s, a, s')$. The belief state summarizes the relevant information about the action–observation history; in particular, any optimal policy for the belief state MDP is also optimal for the POMDP, assuming the belief state is maintained exactly.

2.4 Solution algorithms

The literature has tended to focus on the *offline planning* problem of finding a full policy π over the belief space. Due to the constraints of running on a robot in real-time, we consider instead the *online planning* problem (Ross et al. 2008) where the agent is repeatedly given an observation and just returns an action for the current belief state. Most online algorithms are based on *forward search*. A forward search

tree for (belief state) MDPs consists of alternating layers of action nodes and chance nodes. The root of the tree is an action node labelled with the initial (belief) state. An action node has children corresponding to the possible actions. The chance node corresponding to a state-action has children corresponding to the possible successor states, labelled with the probability of that particular state. A search tree can be used to estimate the value of taking each action at the root by repeated *backups*. The leaf action nodes are given values according to some heuristic estimate of the value function at their state. The value of a chance node is the average of the child values weighted by the probabilities. The value of an action node is the maximum of the child values.

There are various choices in how to generate search trees. First, the children of a chance node can be generated either exhaustively, based on an explicitly given transition model of the MDP, or indirectly, by sampling repeatedly from it, which only requires a simulator. Second, there is the choice of which nodes to expand, given finite total computation time. Apart from simple fixed-depth strategies (Kearns, Mansour, and Ng 1999), there are various more sophisticated methods based on heuristics, branch-and-bound, and meta-level reasoning (Ross et al. 2008; Kocsis and Szepesvári 2006; Russell and Wefald 1991). Finally, the choice of evaluation function at the leaves is of key importance, especially when the tree depth is much lower than the expected time to termination.

3. Problem Formulation

We now consider various formulations of the navigation problem. All of them will be defined with respect to a *topological roadmap* over the static map. This is generated as follows from a known static map: first, waypoints are sampled from the free space. This can be done using simple uniform tiling sampling, or more intelligently, e.g., based on the Voronoi graph of free space (Choset and Burdick 2000). The main constraint is that for every point in free space, there is a path of length under some fixed radius threshold R leading to a waypoint. Next, given the waypoints, we run standard path planning offline on pairs of nearby waypoints to generate edges. If obstacles are not considered, such a graph can be used for planning by adding the start and goal positions to the graph, then searching for a path between them. When the robot is at a waypoint, it does a quick local reachability check to the neighbors in the graph. This process acts as a deterministic virtual sensor (Lavalle 2009) that allows us to know the state of all edges incident to the node of the graph the robot is currently at.

3.1 Deterministic

A simple scheme for updating the roadmap is to maintain a list of blocked edges. We add an edge to this list whenever it is observed blocked and remove it when it is observed free. An immediate problem is that the robot can eventually get into a situation where no path exists. To avoid getting stuck, we unblock all edges that aren't currently observed blocked if we are in a situation where a path cannot be found. If a path still can't be found, a wait action is chosen.

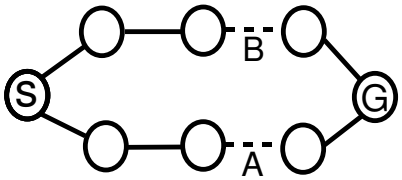

Figure 2:

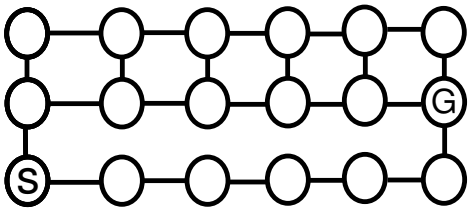

Figure 3:

This scheme does not take any account of the likelihood of an edge being blocked, or of the relative cost of alternative paths. It therefore makes various kinds of systematic errors.

Example 1. *(Block probabilities) In Figure 2, there are two paths, both blocked, but edge A has been observed blocked much more recently than edge B. It therefore makes sense to take the top path.*

Example 2. *(Prediction) In Figure 3, no edges have been observed blocked. It nevertheless makes sense to take the top path, because a single blocked edge on that path can be circumvented, while a single blocked edge on the bottom path requires going back to the start.*

Example 3. *(Patience) In Figure 4, if an obstacle is observed on the edge between S and G, it might make sense (assuming a dynamic model of obstacles) to wait rather than taking the long path, since the obstacle could disappear.*

In each of these cases, the deterministic algorithm will choose the wrong action either always or often.

3.2 Deterministic with blocked-edge costs

Rather than viewing blocked edges as completely impassable, we could give them an extra cost proportional to their

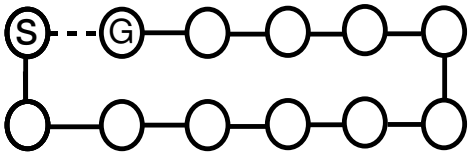

Figure 4:

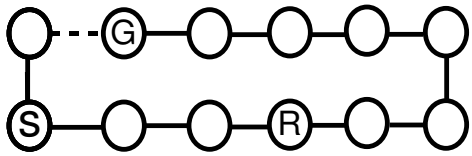

Figure 5:

probability of being blocked. In other words, given an edge such that we last observed it T seconds ago, and it was blocked, its cost is $c(e) + e^{-aT}B$. It therefore deals correctly with Example 1. It still fails on Examples 2 and 3. While this algorithm takes more account of uncertainty, it still neglects the fact that actions can add information.

Example 4. *(Value of information) In Figure 5, suppose the robot is at S, and the short path to G has been recently observed blocked. Given the length of the long path, it might still make sense to check again if the path has become free before deciding on the long path. It is not possible to achieve this in general by adjusting the a and B parameter, for that would prevent the robot from ever considering the long path.*

Having costs change over time can also lead to another problem.

Example 5. *(Persistence) In Figure 5, suppose now that the robot is at R, proceeding down the long path which was chosen because the short path to G from S had a high initial probability of being blocked. The probability of being blocked will decrease exponentially though, so it is possible that the edge's blocked cost decreases enough that the robot will stop in the middle of the path and go back down the other way, which leads to behavior that, apart from being suboptimal, looks strange to humans.*

3.3 Most likely state

A related method is to maintain a distribution over the true state of the world, then plan assuming the most likely graph. As above, this method fails to take sensing actions in Example 4.

3.4 MDP

Rather than using costs, we can model the uncertainty using an MDP. A straightforward way to do this would be to have each edge's status be sampled afresh each time the robot is adjacent to it. The problem this runs into is that, since the edges have no hidden state, a behavior such as "wait for 20 seconds, then choose another path if this edge is still blocked" would never be followed in cases like Example 4 — the robot would either leave immediately or wait forever.

3.5 POMDP

POMDPs model both the transition and sensing uncertainty in the domain. We use the following POMDP model:

- There is one state variable for the current position in the topological roadmap and, for each edge a status, which can either be free or blocked. In the latter case it belongs to one of a predefined set of classes. In our examples, the obstacle classes are `temporary`, `person`, and `static`.
- The actions at a state are to take one of the outgoing edges from that node, or to wait for 1 second at the current position.
- The transition model is that a move succeeds iff the edge is not blocked (at the start of the move). If so, it has a duration depending on the edge length, and the robot ends up at the other node incident to the edge. Additionally, each edge status evolves according to an independent continuous time Markov chain. Note that the despite the action is considered to take one timestep regardless of the duration. This does not affect anything because we do not use discounting.
- The observation model is that we know the current position and, for each adjacent edge, we observe whether it's free or blocked (but not which class the obstacle belongs to).
- The cost of an action is the time it takes.

Optimal solutions to this POMDP avoid the problems listed above. For example, in Figure 5, the belief will include a distribution on whether the obstacle is static, temporary, or a person. If the probability of being temporary or a person is low enough, the optimal plan will be to move to the blocked edge and wait for some amount of time. If the path clears, take it. If not, the probability of being static will eventually increase enough that going the long way becomes optimal.

4. State Estimation

4.1 Belief Update

A belief update given belief b, action a, and observation o, consists of a projection through a followed by conditioning on o. In our case, the transition model over state is:

$$P(s'|a,s) = I_{\{s'_p = f_p(s,a)\}} \prod_{(u,v)\in E} P(s'_{uv}|s_{uv}, t(s,a))$$

where:

- $f_p(s,a)$ is the deterministic transition function of position that results in moving to the other incident node of a if the move is legal, and staying at the current position for 1 second otherwise. Wait actions also result in staying at the current position.
- $t(s,a)$ is the duration of the action, which is just the edge length.
- $P(s'_{uv}|s_{uv},t)$ is given by the continuous time Markov chain transition distribution in (1)

We represent our belief states in factored form, as consisting of a known position b_p, and a distribution b_{uv} over each edge's status. Given such a belief, since the transition model

above factors over terms, each of which depend on one of the belief variables above, the projected distribution $\mathcal{P}(b, a)$ will also have this factored form. Similarly, the observation model is deterministic, and can be written:

$$Z(o|s, a, s') = \prod_{(u,v)\in E} I_{\{o_{uv}=g(s'_{u,v}, s_p)\}}$$

where the function g returns `free` or `blocked` if s_p equals u or v, and `unobserved` otherwise. Since the position is known, the update can once again be done in factored form: for edges adjacent to the current position, if the edge is observed free the conditional distribution is `free` with probability 1, and if it is observed blocked, we make the probability of it being free 0 and reweight the remaining statuses to sum to 1.

4.2 Reducing the Number of Belief Updates

Any forward search algorithm based on the belief-state MDP must perform belief updates. Existing applications have often been in settings where the state and observation spaces are small enough that the belief can be represented as a vector, and belief update is reasonably efficient. As we move into larger, factored state spaces, the cost of belief update becomes an issue. In our domain, we made use of conditional independence to speed up belief update. More generally, Kearns and McAllester (McAllester and Singh 1999) showed that near-optimal behavior can still be achieved if we use the approximate BK belief update algorithm (Boyen and Koller 1998). Nevertheless, given the large number of node expansions, this step can be a bottleneck. We describe here a trick, for reducing the update cost of sampling-based forward search, that we have not previously seen in the literature.

Consider a chance node in a forward search tree, corresponding to doing action a at state s. We generate W samples from the conditional distribution, and create corresponding child nodes, but if two samples lead to the same successor state s', we merge them together and increase the weight of the parent edge. A straightforward implementation would be to check, upon generating a sample s', whether it equals one of the previously generated states.

In a belief-state MDP, given (belief) state b and action a, samples b' are generated by sampling a state s from b, a state s' from $P(\cdot|s, a)$, and an observation o from $Z(\cdot|s, a, s')$, then letting $b' = \mathcal{F}(b, a, o)$. The standard implementation would first compute b' and check for equality with previously generated belief states. Instead, though, we can use the fact that $\mathcal{F}$ is deterministic: along with each successor belief state, we store the observation that led to it. To generate a new sample, we sample s, s', and o as before, then check if observation o has already been sampled (possibly with different s and s'), and only call $\mathcal{F}$ if not.

Theorem 1. *Given that the top k observations in any belief state have total probability at least $1 - \epsilon$, a tree of sampling width W requiring N belief updates without the above technique requires on average $N(\epsilon + k/w)$ belief updates with it.*

5. Efficient Planning

The state space of the POMDP model is exponential in the number of edges, which is beyond the capabilities of current general-purpose offline planners. Online planners based on forward search don't directly depend on the state space size, but do have exponential dependence (for a fixed bound on error) on the search horizon, which can be large in our case, since plan sizes can be on the order of the graph diameter.

5.1 The Abstract Problem

We'll take advantage of structure in the problem. Suppose we are at some belief state, i.e., at some known position with given distributions over the edge statuses. Suppose in particular, that there are K edges that are potentially blocked. If we neglect the possibility of new edges becoming blocked, then an optimal conditional plan will consist of repeatedly visiting (an incident node to) some possibly blocked edge, observing its status, and eventually moving to the goal. So we consider a modified problem where the single edge moves are replaced by *macro-actions* to move to these possibly blocked edges and/or to the goal. The edge costs in the reduced problem are found by solving deterministic shortest path problems, which are also used to find the first edge in the original problem corresponding to an edge in the reduced problem. In the actual problem new obstacles can appear, so we replan after every move (in the original graph) to be able to react to them. Our reduction effectively separates the problem into observation planning and fully observed navigation planning.

More precisely, define the *abstract graph* G_a given a problem and belief state. First, let B be the set of edges whose probability of being blocked exceeds some threshold (we use $\frac{1+p_\beta}{2}$ where p_β is the stationary probability of an unobserved edge being blocked). Let the *cut graph* be the original graph with edges in B removed. Given current position v_s and goal v_g, the abstract graph has vertices $\tilde{V} = V_B \cup \{v_s, v_g\}$ where V_B are the incident vertices to edges in B. For each edge in B, there is a corresponding edge in G_a with the same length. Also, for every pair of vertices $u, v \in \tilde{V}$, if the distance d_{uv} between them in the cut graph is finite, add an edge in G_a between u and v with length d_{uv}. The initial belief b_0 of the abstract problem is the same as b except that edges not in B are not present, and the newly added abstract edges are included, and are considered free with probability 1. The abstract problem has the same unblocking rates as the original problem, but the block rate is 0.

Given the belief state shown in Figure 6, Figure 7 shows the abstract graph and belief state. To emphasize, the abstract problem is still nontrivial to solve because we only observe the incident edges to the current position.

Intuitively, the quality of this approximation depends on how likely new obstacles are to appear.

Theorem 2. *Suppose the edge status Markov chains are such that in any reachable belief state, the optimal cost to the goal is finite. Let β be the block rate, i.e., the probability of an edge blocking in one second. Suppose we start at a belief state with optimal expected cost C, and repeatedly*

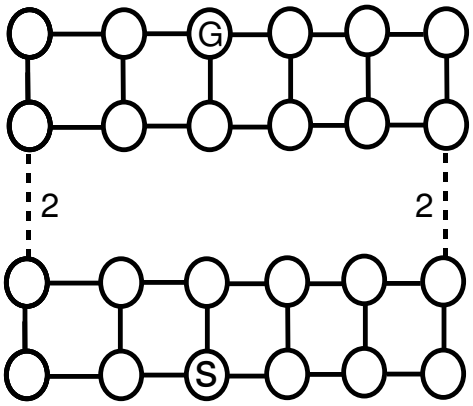

Figure 6: Original problem. Dashed edges have length 2 and have block probability above the threshold. All other edges have length 1, and have block probability below the threshold.

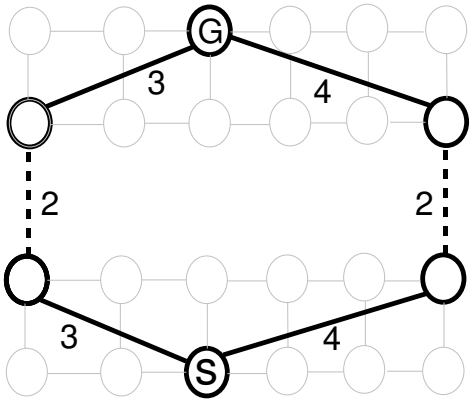

Figure 7: Abstracted version of problem in Figure 6. In the initial belief state, dashed edges are possibly blocked, and all others are definitely free.

do the action resulting from solving the abstract problem exactly. The expected time to the goal is then $C \cdot O(\frac{1}{1-\beta})^D$ *for a constant* D.

So when $\beta = 0$ the abstraction is exact, and in general the abstract policy is guaranteed to do well when β is not too large, which seems like a reasonable assumption in practice. As β gets large, the algorithm will make systematic mistakes in situations like Example 2, where it will fail to take into account what happens when edges become blocked in future.

5.2 Solving the Abstract Problem

The reduced POMDP still has a large state space, but a shorter horizon, making it a good candidate for forward search. We use simple fixed-depth search with sampling. Better lookahead schemes could be used, and would likely improve performance. We evaluate leaves of the search tree using the following heuristic function. Given a belief state b, sample some number of graphs (we generate 100 samples using a variation of the duplicate checking trick from Section 4.2); in each graph, compute the minimum of the shortest path distance to the goal and some threshold (we use the graph diameter), and average the results. Also, since the position and statuses of adjacent edges are known at each belief state, we know for each node in the tree the set of successor actions, and whether or not we have reached the goal.

6. Experiments

We compare several of the approaches described in Section 3. There are two metrics of interest: planning time and execution time in the world. For the case of robotic navigation, where moving between successive nodes in a roadmap typically takes seconds or tens of seconds, the main requirement on the planning time is that an action should be returned within this time. We placed a limit of one second of planning time per action, and compared the following approaches:

- DA is the deterministic agent from Section 3.1.
- BA1 and BA2 are the block-cost agents from Section 3.2, using block costs of 10 and 1000.
- ALA1 through ALA3 are abstract POMDP agents from Section 3.5, using search depths 1 to 3, and a sampling width of 100.

We used a set of domains with graph sizes ranging from 15 to 1000. The smaller ones were generated by hand while the larger ones were random (independent edge probabilities, additional edges were added if they weren't connected). The parameters of the edge status chains were assumed known, though in principle they could be learnt by passive observation. A block rate of .01 was used, and the edge lengths were between 1 and 30. Thus, a typical path would expect to encounter several blocked edges. The planners were run on a standard dual-core 2.66Ghz machine.

The POMDP-based algorithms are the best by a significant margin on each domain. Interestingly, ALA2 occasionally does better than ALA3. This phenomenon is known in the search literature as a lookahead pathology (Bulitko and

Instance	DA	ALA1	ALA2	ALA3	BA1	BA2
1	161 ± 23	186 ± 36	$\mathbf{132 \pm 7}$	142 ± 12	272 ± 57	285 ± 94
2	758 ± 207	225 ± 59	228 ± 65	$\mathbf{149 \pm 38}$	522 ± 207	380 ± 123
3	1203 ± 64	936 ± 58	$\mathbf{907 \pm 54}$	1025 ± 48	1064 ± 73	1134 ± 63
4	353 ± 14	361 ± 33	191 ± 22	$\mathbf{178 \pm 12}$	259 ± 33	286 ± 40
5	897 ± 97	853 ± 122	639 ± 43	$\mathbf{625 \pm 37}$	914 ± 103	1321 ± 91
6	1115 ± 80	1026 ± 56	$\mathbf{915 \pm 54}$	924 ± 51	1079 ± 88	1226 ± 76
7	441 ± 39	493 ± 58	410 ± 44	$\mathbf{378 \pm 36}$	408 ± 37	461 ± 36
8	732 ± 101	857 ± 122	684 ± 55	$\mathbf{621 \pm 51}$	817 ± 62	808 ± 53

Table 1: Results on evaluating different agents on a set of benchmark environments. Each cell entry reports average cost, rounded to the nearest integer, till reaching the goal and sample standard deviation, given 30 independent trials (the randomness is due to the environment uncertainty as well as randomness in the algorithms).

Lustrek 2006). A more intelligent search strategy than uniform depth first search should help to mitigate this problem.

7. Related Work

There is an extensive literature on simultaneous localization and mapping, though mostly focussed on the passive sensing case. (LaValle 2004) discusses active variants, as well as algorithms for representing uncertainty in discrete grids and continuous worlds using reduced information spaces. Our algorithm differs in that it is primarily concerned with navigation rather than building a map, explicitly models changes in the obstacle set, and is based on a graph rather than a metric map.

Navigation using topological roadmaps has been well studied. Most existing work has assumed the graph is known. (Missiuro and Roy 2006) is an exception, in which the obstacles are assumed to be polygons whose vertex locations are uncertain with Gaussian distributions. They modify a probabilistic roadmap motion planner to accept or reject sampled points based on the probability of collision, and then essentially look for an unconditional plan with a high chance of success. In contrast, since our approach has an observation model, it will return plans that condition future actions on the results of observations.

Ong et al (Ong et al. 2009) studied *mixed observability Markov decision processes* (MOMDPs), in which part of the state is perfectly observed, and showed how to speed up offline dynamic programming algorithms in this case. Our problem is a MOMDP, since the robot position in the roadmap is perfectly observed. Unfortunately, the unobserved portion of the state space is still exponentially large.

Theocharous and Kaelbling (Theocharous and Kaelbling 2003) also describe a macro-action-based POMDP planning algorithm for robot navigation. In their setting, the robot position is the only hidden variable and observations give incomplete information about position. They describe an algorithm based on an adaptive discretization of the belief simplex that estimates the entire Q-function (as opposed to just the value from a single state). In our setting, the state space is exponentially larger due to the obstacle states, making full planning on the belief simplex challenging. Making the true position partially observable in our setting could be a useful extension in environments where accurate localization is not possible.

Closely related to this research is that of Kneebone and Dearden (Kneebone and Dearden 2009). Like us, they consider navigation in a partially observed graph, and represent the uncertainty using a POMDP. A major difference is that in their framework, obstacles are static: they do not appear and disappear. This makes a qualitative difference in the kinds of policies that are found: there is no longer any utility in taking wait actions, nor are there useful inferences to be made about obstacle classes (effectively, all obstacles belong to a single class whose unblocking rate is 0). The fact that new obstacles don't appear also allows them to perform an exact reduction of the state space, in which the robot may only be at possible obstacle locations. The above comments also apply to the literature on the Canadian Traveller's Problem (Papadimitriou and Yannakakis 1991).

A complementary line of research is on motion planning among movable obstacles (Hsu et al. 2002). Unlike our work, which considers an unknown and changing set of static obstacles, this work considers a known, fixed set of (stochastically) moving obstacles. A natural extension would be to allow the moving obstacles to appear and disappear, and to have hidden state similar to our obstacle classes, that governs whether and how they move. The problem would be complicated by the need to perform data association.

8. Conclusion

We have presented a formulation of navigation in a changing and partially observable world as a POMDP. This formulation deals with uncertainty in a principled way, and optimal solutions to it exhibit various behaviors that are normally hardcoded in, such as persistence and inference about obstacles. We also described an efficient approximate solution algorithm that can feasibly be run online on a robot, and showed improved performance compared with traditional deterministic planners. Future work includes improving the search control, extending to domains with movable objects, and allowing the robot to take actions that affect the graph connectivity.

References

Boyen, X., and Koller, D. 1998. Tractable inference for complex stochastic processes. In Cooper, G. F., and Moral, S., eds., *UAI*, 33–42. Morgan Kaufmann.

Bulitko, V., and Lustrek, M. 2006. Lookahead pathology in real-time path-finding. In *AAAI*. AAAI Press.

Choset, H., and Burdick, J. W. 2000. Sensor-based exploration: The hierarchical generalized voronoi graph. *I. J. Robotic Res.* 19(2):96–125.

Hsu, D.; Kindel, R.; Latombe, J.-C.; and Rock, S. M. 2002. Randomized kinodynamic motion planning with moving obstacles. *I. J. Robotic Res.* 21(3):233–256.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artif. Intell.* 101(1-2):99–134.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 1999. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, 1324–1231.

Kneebone, M., and Dearden, R. 2009. Navigation planning in probabilistic roadmaps with uncertainty. In Gerevini, A.; Howe, A. E.; Cesta, A.; and Refanidis, I., eds., *ICAPS*. AAAI.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *ECML*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer.

LaValle, S. M. 2004. Planning algorithms.

Lavalle, S. 2009. Filtering and planning in information spaces. *IROS tutorial notes*.

McAllester, D. A., and Singh, S. P. 1999. Approximate planning for factored pomdps using belief state simplification. In Laskey, K. B., and Prade, H., eds., *UAI*, 409–416. Morgan Kaufmann.

Missiuro, P. E., and Roy, N. 2006. Adapting probabilistic roadmaps to handle uncertain maps. In *ICRA*, 1261–1267. IEEE.

Norris, J. R. 1999. *Markov Chains*. Cambridge University Press, 1 edition.

Ong, S.; Png, S.; Hsu, D.; and Lee, W. 2009. POMDPs for robotic tasks with mixed observability. In *Proc. Robotics: Science and Systems*.

Papadimitriou, C. H., and Yannakakis, M. 1991. Shortest paths without a map. *Theor. Comput. Sci.* 84(1):127–150.

Puterman, M. 2005. *Markov decision processes*. Wiley-Interscience.

Ross, S.; Pineau, J.; Paquet, S.; and Chaib-draa, B. 2008. Online planning algorithms for pomdps. *J. Artif. Intell. Res. (JAIR)* 32:663–704.

Russell, S. J., and Wefald, E. 1991. Principles of metareasoning. *Artif. Intell.* 49(1-3):361–395.

Theocharous, G., and Kaelbling, L. P. 2003. Approximate planning in pomdps with macro-actions. In Thrun, S.; Saul, L. K.; and Schölkopf, B., eds., *NIPS*. MIT Press.