Anytime Path Planning and Replanning in Dynamic Environments

Jur van den Berg
Department of Information and Computing Sciences
Universiteit Utrecht
The Netherlands
berg@cs.uu.nl

Dave Ferguson and James Kuffner
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania
{davef,kuffner}@cs.cmu.edu

***Abstract*— We present an efficient, anytime method for path planning in dynamic environments. Current approaches to planning in such domains either assume that the environment is static and replan when changes are observed, or assume that the dynamics of the environment are perfectly known a priori. Our approach takes into account all prior information about both the static and dynamic elements of the environment, and efficiently updates the solution when changes to either are observed. As a result, it is well suited to robotic path planning in known or unknown environments in which there are mobile objects, agents or adversaries.**

I. Introduction

A common task in robotics is to plan a trajectory from an initial robot configuration to a desired goal configuration. Depending on the nature of the problem, we may be interested in *any* collision-free trajectory, or one that provides the minimum (or close to minimum) overall cost, where the cost of a trajectory may be a function of several factors including time for traversal, traversal risk, stealth, and visibility.

Several approaches exist for generating such trajectories. A popular technique in mobile robot navigation is to uniformly discretize the configuration space and then plan an optimal trajectory through this discretized representation using Dijkstra's algorithm or A* [1]. However, for more complex configuration spaces, such as those involving robots with several degrees of freedom, using a uniform discretization of the configuration space is intractable in terms of both memory and computation time. As a result, randomized approaches such as Rapidly-exploring Random Trees (RRTs) and Probabilistic Roadmaps (PRMs) have been widely-used in these domains [2], [3]. These approaches work by randomly sampling points in the continuous configuration space and then growing the current solution out towards these points.

Extensions have also been developed to the above approaches that efficiently repair the current solution when changes are made to the configuration space [4], [5], [6], [7]. These considerations are particularly important when we are dealing with a sensor-equipped robotic agent moving through a partially-known environment, as the agent will be continually updating its information through its sensors.

When there are dynamic elements in the environment (e.g. moving obstacles or adversaries), the planning problem is more challenging. There are again several approaches that can be taken. First, we can assume the environment is static and use any of the approaches above, then when changes are observed (due to the dynamic elements) we can replan. This can be efficient, but does not take into account any information the agent may have concerning the dynamic elements (for instance, their velocities and directions of movement) and hence may produce highly suboptimal results.

There is a qualitative difference between dynamic and static planning problems that these approaches do not address. For instance, imagine an agent trying to cross a road on which cars are driving. If the agent was to take a snapshot of the environment with its sensors and assume fixed positions for each object, then it would be in serious risk of getting runover if it attempted to cross the road. In order to successfully accomplish its task, the agent really needs to model the cars as dynamic objects so that it can anticipate where they will be at future times.

A second set of approaches does just this, by planning in the joint configuration-time state space. In such a situation, the trajectories of dynamic objects can be modelled explicitly and taken into account when performing initial planning. Time-optimal or near time-optimal approaches for computing paths through state-time space have been developed [8], however these algorithms are typically limited to low dimensional state spaces and/or require significant computation time. In [9], [10], probabilistic approaches are used for computing paths in state-time space. These planners incrementally build a tree of explored configurations for each planning query.

More recently, researchers have looked at reducing the complexity of the planning problem by first constructing a path [11] or a PRM [12] based on the static elements of the environment, and subsequently planning a collision-free trajectory on this path/roadmap that takes into account the dynamic obstacles.

The combination of probabilistic sampling and deterministic planning has been found to be particularly useful in generating time-optimal trajectories through roadmaps with known dynamic elements. However, none of the approaches mentioned above efficiently solves the general case, where an agent may have (i) perfect information, (ii) imperfect information, *and/or* (iii) no information, regarding the dynamic elements of the environment, and the agent needs to quickly repair

its trajectory when new information is received concerning *either* the dynamic *or* static elements of the environment. Further, most of these approaches assume that the only factor influencing the overall quality of a solution is the time taken for the agent to traverse the resulting path. In fact, we are often concerned with minimizing a more general cost associated with the path that may incorporate, for example, traversal risk, stealth, and visibility, as well as time of traverse.

In this work, our goal is to combine some of the positive characteristics of several previous algorithms with new ideas to generate an approach that provides an effective solution to the general problem of planning low-cost trajectories in dynamic environments. Our approach uses probabilistic sampling to create a robust roadmap encoding the planning space, then plans and replans over this graph in an anytime, deterministic fashion. The resulting algorithm is able to generate and repair solutions very efficiently, and improve the quality of these solutions as deliberation time allows.

We begin by describing the problem we are trying to solve. In Section III we introduce our new algorithm and describe how it relates to current approaches. We present a number of results in Section IV. We conclude with discussion and extensions.

II. Problem Description

Consider a mobile robot navigating a complex, outdoor environment in which adversaries or other agents exist (such as the one presented in Fig. 1). Imagine that this environment contains terrain of varying degrees of traversability, as well as desirable and undesirable areas based on other metrics (such as proximity to adversaries or friendly agents, communication access, or resources). Imagine further that this agent is trying to reach some goal location in the environment, while incurring the minimum possible combined cost according to all of the above metrics, as well as (perhaps) time. Specifically, imagine the agent is trying to minimize the cost of its trajectory according to some function $\mathcal{C}$, defined as:

$$\mathcal{C}(\text{path}) = w_t \cdot t_{\text{path}} + w_c \cdot c_{\text{path}},$$

where t_{path} is the time taken to traverse the path, c_{path} is the cost of the path based on all relevant metrics other than time, and w_t and w_c are the weight factors specifying the extent to which each of these values contributes to the overall cost of the path ($w_t + w_c = 1$).

In a real world scenario it is likely that the agent will not have perfect information concerning the environment, and there may be dynamic elements in the environment that are not under the control of the agent. It is important that the agent is able to plan given various degrees of uncertainty regarding both the static and dynamic elements of the environment.

Further, as the agent navigates through this environment, it receives updated information through onboard sensors concerning both the environment itself and the dynamic elements within. Thus, the planning problem is continually changing and it is important that the agent can repair its previous solution or generate a new one to account for the latest

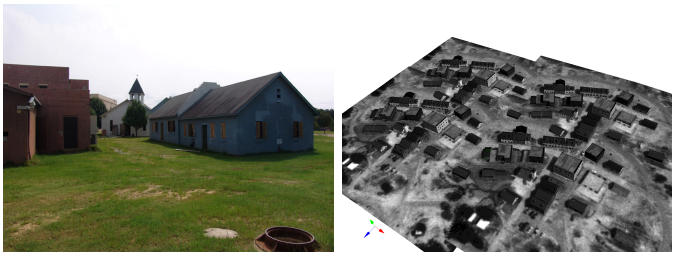

Fig. 1. Data acquired from Fort Benning and used for testing.

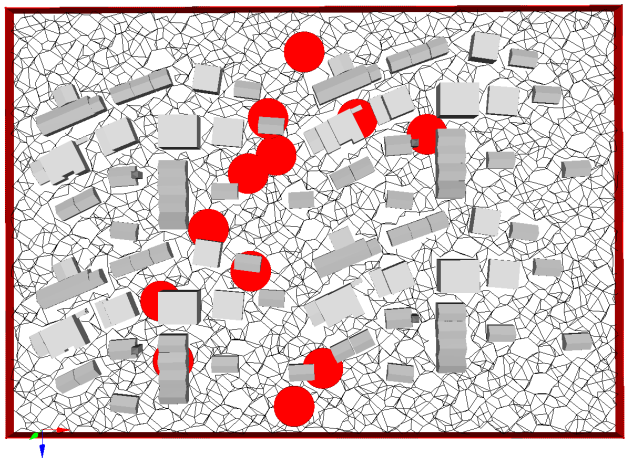

Fig. 2. A roadmap overlaid onto the Fort Benning data, along with 12 dynamic obstacles (in red).

information. This planning must be performed in a very timely manner, as the best solutions may require immediate action and so that solutions are not out-of-date when they are finally generated.

Our aim in this work is to do just this by combining the efficient replanning ability of deterministic incremental planning algorithms with the efficient representations produced by probabilistic sampling approaches. Further, we would like our resulting approach to be anytime, so that solutions can be generated very quickly when deliberation time is limited, and these solutions can be improved while deliberation time allows.

III. Approach

Our approach consists of three separate stages. In the first, a roadmap is built representing the static portion of the planning space. Next, an initial trajectory is planned over this roadmap in state-time space, taking into account any known dynamic obstacles. This trajectory is planned in an anytime fashion and is continually improved until the time for deliberation is exhausted. Further, while the agent executes its traverse, its trajectory continues to be improved upon. Finally, when changes are observed, either to the static or dynamic elements of the environment, the current trajectory is repaired to reflect these changes. Again, this is done in an anytime fashion, so that at any time a solution can be extracted.

A. Constructing the roadmap

As in other recent approaches dealing with dynamic environments (e.g. [12], [7]), we begin by creating a PRM taking into account the static portion of the environment. This PRM encodes any internal constraints placed on the robot's motion (such as degrees of freedom, kinematic limitations, etc) and takes into account the known costs associated with traversing different areas of the environment. It also includes cycles to allow for many alternative routes to the goal [13]. The objective in this initial phase is to reduce the continuous planning space into a discrete graph that is compact enough to be planned over while still being extensive enough to provide low-cost paths. Fig. 2 illustrates a PRM constructed from the outdoor environment presented in Fig. 1.

B. Planning over the roadmap

We then plan a path over this PRM from the agent's initial location to its goal location, taking into account any known dynamic elements. To do this, we add the time dimension to our search space, so that each state s consists of a node on the PRM n and a time t. This allows us to represent trajectories of dynamic elements. We discretize the time-axis into small steps of δ_t, and allow transitions from state (n, t) to state $(n, t + \delta_t)$ and to states $(n', t + c_t(n, n'))$, where n' is a successor of n in the roadmap and $c_t(n, n')$ is the time it takes to traverse the edge between them. This allows the robot to wait at a particular location as well as to transition to an adjacent roadmap location. The total cost of transitioning between state $s = (n, t)$ and some successor $s' = (n', t')$ is defined as:

$$c(s, s') = w_t \cdot (t' - t) + w_c \cdot c_r(n, n'),$$

where $c_r(n, n')$ is the cost of traversing the edge between n and n' in the roadmap.

Because the robot may not have perfect information concerning the dynamic elements in the environment, it is important that it adequately copes with partial information. There are a number of existing methods for dealing with this scenario [14], [10]. In particular, we can estimate future trajectories based on current behavior, or we can assume worst-case trajectories. Whichever of these we choose, we end up with some trajectory or set of trajectories that we can represent as 3D objects in our state-time space (see Fig. 3). We can then avoid these objects as we plan a trajectory for the agent.

Planning a least-cost path through this space can be computationally expensive, and although the agent may have time to generate its initial path, if the agent is continuously receiving new information as it moves, replanning least-cost paths over and over again from scratch may be infeasible.

Instead, it would be much better if the agent could repair its previous solution incrementally, e.g. as in D* and D* Lite [4], [5]. In order to do this, it needs to plan in a backwards direction from the goal to the start, so that when the agent moves, the stored paths and path costs of all the states in the search space that have already been computed are not

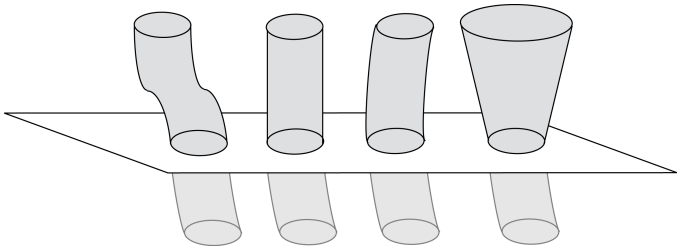

Fig. 3. Different dynamic obstacles based on information. On the far left is a known trajectory, on the center-left an assumed-static obstacle, on the center-right an extrapolated trajectory based on previous motion, and on the far right a worst-case trajectory based on current position and maximum velocity.

affected[1]. Since we don't know in advance at what time the goal will be reached, we seed the search with multiple goal states. For our implementation,

$$\begin{aligned} \textit{GOALS} = [&(n_{\text{goal}}, h_t(n_{\text{start}}, n_{\text{goal}})), \\ &(n_{\text{goal}}, h_t(n_{\text{start}}, n_{\text{goal}}) + \delta_t), \\ &\vdots \\ &(n_{\text{goal}}, \textit{max-arrival-time})] \end{aligned}$$

where n_{goal} is the goal node in the PRM, *max-arrival-time* is the maximum time allowed for traveling to the goal, and h_t is described below.

To improve the efficiency of the search, we use two important heuristic values. First, we compute the minimum possible *time* $h_t(n_{\text{start}}, n)$ for traversing from the current start position n_{start} to any particular position n in the PRM. Second, we compute the minimum possible *cost* $h_c(n_{\text{start}}, n)$ from the current start position to any particular position n on the PRM. We then use the time heuristic to prune states added to the search and the cost heuristic to focus the search.

Specifically, if we are searching backwards from the goals and come across some state $s = (n_s, t_s)$, then if $t_s - t_{\text{start}} < h_t(n_{\text{start}}, n_s)$ we know that it is not possible to plan a trajectory from the initial location and initial time to this location by time t_s, so this state cannot be part of a solution and can be ignored. If the state passes this test, then we insert it into our search queue with a priority based on a heuristic estimate of the overall cost of a path through this state:

$$\text{key}(s) = rhs(s) + w_t \cdot (t_s - t_{\text{start}}) + w_c \cdot h_c(n_{\text{start}}, n_s),$$

where $rhs(s)$ is the current cost-to-goal value of state s. This overall heuristic estimate serves the same purpose as the f-value in classic A* search: it focuses the search towards the most relevant areas of the search space.

However, as already mentioned, the agent may not have time to plan and replan optimal paths across the PRM. Instead, it may need to be satisfied with the best solutions that can be generated in the time available. To this end, we make use of a recently developed algorithm, *Anytime D** (AD*), that incrementally plans and replans in an anytime fashion [15].

[1]If we store a cost-to-goal (as in D*) rather than a cost-to-start (as in A*), then when the robot state (start) changes, the costs are still valid, as the goal to which the costs refer remains unchanged.

AD* begins by quickly generating an initial, highly-suboptimal solution, by inflating the heuristic value for each state by some $\epsilon > 1$. It then works on efficiently improving this solution while deliberation time allows, by decreasing ϵ. At any point during its processing, the current solution can be extracted and traversed. Then, as the agent begins its execution, AD* is able to continually improve the solution.

key(s)
1 if ($g(s) \geq rhs(s)$)
2 return $[rhs(s) + \epsilon \cdot (w_t \cdot (t_s - t_{\text{start}}) + w_c \cdot h_c(n_{\text{start}}, n_s)); rhs(s)]$;
3 else
4 return $[g(s) + w_t \cdot (t_s - t_{\text{start}}) + w_c \cdot h_c(n_{\text{start}}, n_s); g(s)]$;

UpdateSetMembership(s)
5 if ($g(s) \neq rhs(s)$)
6 if ($s \notin$ *CLOSED*) insert/update s in *OPEN* with key(s);
7 else if ($s \notin$ *INCONS*) insert s into *INCONS*;
8 else
9 if ($s \in$ *OPEN*) remove s from *OPEN*;
10 else if ($s \in$ *INCONS*) remove s from *INCONS*;

ComputePath()
11 while ($\min_{s \in OPEN}$(key(s)) $\dot{<}$ key(s_{start}) OR $rhs(s_{\text{start}}) > g(s_{\text{start}})$)
12 remove state s with the smallest $key(s)$ from *OPEN*;
13 if ($g(s) > rhs(s)$)
14 $g(s) = rhs(s)$; *CLOSED* = *CLOSED* $\cup \{s\}$;
15 for each predecessor s' of s
16 if s' was not visited before
17 $g(s') = rhs(s') = \infty$; $bp(s') = \textbf{null}$;
18 if ($rhs(s') > g(s) + c(s', s)$)
19 $bp(s') = s$;
20 $rhs(s') = g(s) + c(s', s)$; UpdateSetMembership($s'$);
21 else
22 $g(s) = \infty$; UpdateSetMembership(s);
23 for each predecessor s' of s
24 if s' was not visited before
25 $g(s') = rhs(s') = \infty$; $bp(s') = \textbf{null}$;
26 if ($bp(s') = s$)
27 $bp(s') = \text{argmin}_{s'' \in Succ(s')}(g(s'') + c(s', s''))$;
28 $rhs(s') = g(bp(s')) + c(s', bp(s'))$; UpdateSetMembership($s'$);

Fig. 4. ComputePath function in AD*

We have included pseudocode of our approach, along with AD*, in Figs. 4 through 6.

C. Repairing the Plan

While the agent is traveling through the environment, it will be receiving updated information regarding its surroundings through its onboard sensors. As a result, its current solution trajectory may be invalidated due to this new information. However, it would be prohibitively expensive to replan a new trajectory from scratch every time new information arrives.

Instead, our approach is able to repair the previous solution, in the same way as incremental replanning algorithms such as D* and D* Lite. However, as with initial planning, it is also able to do this repair in an anytime fashion. Thus, solutions can be improved and repaired at the same time, allowing for true interleaving of planning, execution, and observation.

Main()
1 construct PRM of static portion of environment;
2 use PRM and Dijkstra's to extract predecessor and successor functions, heuristic functions h_c and h_t, and goal list *GOALS*;
3 $g(s_{\text{start}}) = rhs(s_{\text{start}}) = \infty$;
4 *OPEN* = *CLOSED* = *INCONS* = $\emptyset$; $\epsilon = \epsilon_0$;
5 for each s_{goal} in *GOALS*
6 $g(s_{\text{goal}}) = \infty$; $rhs(s_{\text{goal}}) = 0$; $bp(s_{\text{goal}}) = \textbf{null}$;
7 insert s_{goal} into *OPEN* with key(s_{goal});
8 fork(MoveAgent());
9 while ($s_{\text{start}} \notin$ *GOALS*)
10 ComputePath();
11 publish ϵ-suboptimal solution path;
12 if ($\epsilon = 1$) wait for changes in edge costs;
13 for all directed edges (s, s') with changed edge costs
14 $c_{\text{old}} = c(s, s')$; update the edge cost $c(s, s')$;
15 if $s \notin$ *GOALS*
16 if s was not visited before
17 $g(s) = rhs(s) = \infty$; $bp(s) = \textbf{null}$;
18 if ($c(s, s') > c_{\text{old}}$ AND $bp(s) = s'$)
19 $bp(s) = \arg\min_{s'' \in Succ(s)} g(s'') + c(s, s'')$;
20 $rhs(s) = g(bp(s)) + c(s, bp(s))$; UpdateSetMembership($s$);
21 else if ($rhs(s) > g(s) + c(s, s')$)
22 $bp(s) = s'$;
23 $rhs(s) = g(s') + c(s, s')$; UpdateSetMembership($s$);
24 if significant edge cost changes were observed
25 increase ϵ or re-plan from scratch;
26 else if ($\epsilon > 1$) decrease ϵ;
27 Move states from *INCONS* into *OPEN*;
28 Update the priorities for all $s \in$ *OPEN* according to key(s);
29 *CLOSED* = $\emptyset$;

Fig. 5. Main function in AD*

MoveAgent()
1 while ($s_{\text{start}} \notin$ *GOALS*)
2 update s_{start} to be successor of s_{start} in current solution path;
3 use Dijkstra's to recompute h_c and h_t given the new value of s_{start};
4 while agent is not at s_{start}
5 move agent towards s_{start};
6 if new information is received concerning the static environment
7 update the affected edges in the PRM;
8 update the successor and predecessor functions;
9 use Dijkstra's to recompute h_c and h_t;
10 mark the affected edges in the AD* search as changed
11 (so that these edges are triggered in Fig. 5 line 14);
12 if new information is received concerning the dynamic elements
13 mark the affected edges in the AD* search as changed
14 (so that these edges are triggered in Fig. 5 line 14);

Fig. 6. Agent Traverse Function

To do this, it first updates the heuristic values of states on the roadmap based on the current position of the robot. This can be done very quickly as it only concerns the static roadmap. It then finds the states in the search tree that have been affected by the new information and updates these states. As we discuss in the following section, this can also be performed efficiently.

IV. Experiments and Results

A. Implementation Details

As said in the previous section, when the agent observes changes regarding the trajectory of the dynamic obstacles, we

should first find all edges in state-time space considered so far whose collision status must be updated. This can potentially be rather expensive, but if we model both the agent and the dynamic obstacles as discs moving in the plane, we can do this very efficiently. First, we add the radius of the agent to the radii of each obstacle, so that we can treat the agent as a point. Next, we mathematically describe the state-time volumes carved out by each dynamic obstacle. For instance, if the estimated trajectory of a dynamic obstacle is an extrapolation of its current velocity, this volume becomes a slanted cylinder, which can be described as

$$((x - x_0) - (t - t_0)v_x)^2 + ((y - y_0) - (t - t_0)v_y)^2 = r^2,$$

where (x_0, y_0) is the current position of the obstacle, (v_x, v_y) its current velocity, r its radius, and t_0 the current time. If we assume the obstacles to be static, this volume becomes a vertical cylinder, and if we model worst-case trajectories, this volume becomes a cone (provided that the maximal velocity of the obstacle is given). In any case, it is easily checked whether or not an edge in state-time space intersects these volumes. Experiments show that it is also very fast; over 50000 edges can be checked within 0.01 seconds.

As the environment is dynamic, these estimated future trajectories may change over time. For instance, if we use the extrapolation method, whenever an obstacle changes its velocity we should re-check all edges against its new volume. However, given an indication of the frequency of trajectory changes (the dynamicity of the environment) we can set some *horizon* on the validity of the estimated trajectory such that only edges in the near future are collision-checked. Edges in the far future are simply considered to be collision-free. As time goes by, these edges are eventually checked as well. This facilitates replanning as collision-checks become faster and less state-time space becomes (unnecessarily) inaccessible when searching for a path.

B. Experimental Setup

We experimented with our method in the environment of Figs. 1 and 2. The agent (a point) has to move from the lower-rightmost node in the roadmap to the upper-leftmost node. The roadmap consists of 4000 vertices and 6877 edges. Twelve disc-shaped dynamic obstacles (see Fig. 2) start in the centre of the scene and spread out in random directions with random velocities. During the traversal of the path by the agent, the obstacles may change their course. These changes are randomly generated and are not known by the agent. In our experiments the total number of course changes varies around 100. Moreover, when an obstacle hits the boundary of the scene, it is bounced backwards, which accounts for additional course changes. The obstacles heavily impede the agent in its attempt to reach the goal node. For the sake of experimentation, the cost values for traversing edges in the roadmap were chosen to be random variations of their length. The time axis was discretized into intervals of 0.1 seconds.

The initial value of ϵ is 10, and this is gradually decreased to 1 as deliberation time allows. After every 0.1 seconds, we check whether the collision status of any edge in state-time space encountered thus far has changed with respect to the obstacles. Each time this occurs, ϵ is reset to 10 to quickly repair the path. Meanwhile, the position of the agent along its path is updated every 0.1 seconds according to the best available path. This continues until the agent has reached the goal. The experiments were performed on a Pentium IV 3.0GHz with 1 Gbyte of memory.

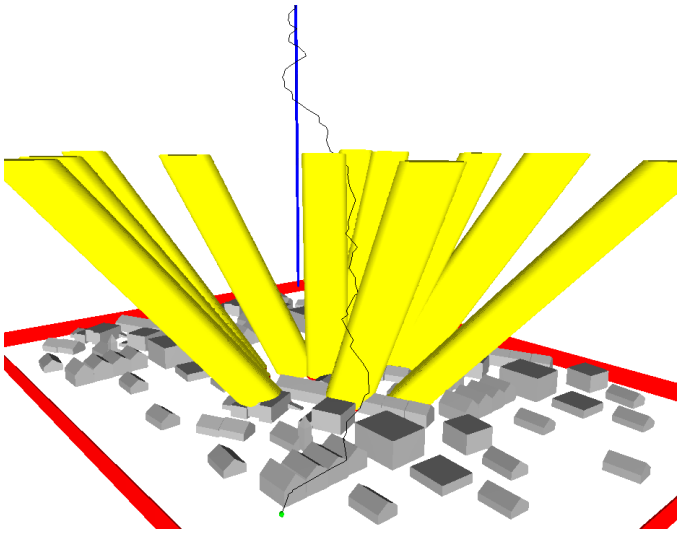

Fig. 7. An example path (in black) planned through state-time space from the initial robot position (in green) to the goal (shown as a vertical blue line extending from the goal location upwards through time). Also shown are the extrapolated trajectories of the dynamic obstacles (in yellow). The underlying PRM has been left out for clarity but can be seen in Fig. 2.

In our experiments we used two models for estimating the future course of the obstacles: the extrapolation method and the assumed-static method (see Fig. 3). In the extrapolation method we assume that the current course of an obstacle (a linear motion) is also its future course. This gives a slanted cylindrical obstacle in state-time space, which is avoided during planning. In the assumed-static case, we assume the obstacles to be static (analogous to several previous approaches), giving vertical cylindrical state-time obstacles. In each iteration of the algorithm the position of the obstacle is updated according to its actual trajectory. For both models the horizon was set to 10 seconds. We also implemented the worst-case model (assuming a maximum velocity for each obstacle), but this approach was not useful for this problem, as the obstacles could move so quickly that their worst-case conical volumes quickly grew so large that no feasible paths existed. Fig. 7 shows an example path planned through state-time space, along with the extrapolated trajectories of the dynamic obstacles.

C. Results

In our experiments we compared the PRM-based Anytime D* method (in which ϵ is regularly reset to 10) to a PRM-based D* Lite method (in which ϵ is always 1). The Anytime D* method was run in real-time, so that the agent was moved along its current path regardless of whether it had finished

TABLE I

RESULTS (AVERAGED OVER 50 RUNS)

Obstacle Model	Approach	Cost	Max.time	Invalid
Extrapolation	Anytime D*	81.07	0.19s	22%
Extrapolation	D* Lite	80.20	0.74s	18%
Assumed-Static	Anytime D*	85.09	0.22s	52%
Assumed-Static	D* Lite	81.08	0.67s	58%

repairing or improving the path. The D* Lite method was not run in real-time and the agent was allowed as much time for planning as it needed in between movements (in other words, time was 'paused' for the planner). We included this latter approach simply to demonstrate the relative efficiency and solution quality of Anytime D* compared to an optimal planner.

For each run (in which the path is improved and repaired many times) we measured the maximum time needed to improve or repair the path. As the obstacle trajectories are randomly generated, we performed 50 runs using each method with the same random sequence, and averaged the maximal planning times.

In addition, we measured the average overall cost of the path, and the number of times the generated path was not safe (i.e. in which there was a collision between the agent and the obstacles). Results are reported for both the extrapolation model and the static model in Table I.

From the results we can see that for both models the maximal amount of time needed to replan the path is on average three to four times less for Anytime D* than for D* Lite. The path quality did not suffer much from the anytime-characteristic: for both Anytime D* and D* Lite the average path costs are about the same.

If we compare the extrapolation model (in which we use information of the current velocity of the obstacles) to the assumed-static case, we see that the assumed static case is not very safe. In approximately half of the cases, the agent hits an obstacle. In the extrapolation case about one fifth of the runs result in a collision. These collisions are explained by the radical course changes the obstacles can make; if the agent moves alongside an obstacle, and suddenly the obstacle decides to take a sharp turn, the agent may not have enough time to escape a collision (also because the velocity of the obstacle is unbounded). We expect that more realistic obstacle behavior in combination with slightly more conservative trajectory estimation will remedy this.

D. Extensions

It may be possible to make this approach even more efficient by using an expanding horizon for the dynamic obstacles. Specifically, when producing an initial plan we set the horizon for each obstacle to zero, so that a path is produced very quickly. Then, as deliberation time allows, we improve the accuracy and robustness of this path by gradually increasing the horizon. This modification may significantly reduce the time required to generate an initial path.

V. DISCUSSION

We have presented an approach for anytime path planning and replanning in partially-known, dynamic environments. Our approach handles the case where an agent may have (i) perfect information, (ii) imperfect information, *and/or* (iii) no information, regarding the dynamic elements of the environment, and the agent needs to quickly repair its trajectory when new information is received concerning *either* the dynamic *or* static elements of the environment. We have shown it to be capable of solving large instances of the navigation problem in dynamic, non-uniform cost environments.

Our approach combines research in deterministic and probabilistic path planning. We are unaware of any approaches to date that combine the strong body of work on deterministic replanning algorithms with compact, probabilistically-generated representations of the environment, and yet we believe the union of these two areas of research can lead to very effective solutions to a wide range of problems. As such, we are currently looking at how some of these ideas can be applied to other robotics domains.

REFERENCES

[1] N. Nilsson, *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[2] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[3] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[5] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[6] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[7] L. Jaillet and T. Simeon, "A PRM-based motion planner for dynamically changing environments," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[8] J. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer Academic Publishers, 1991.

[9] D. Hsu, R. Kindel, J. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[10] S. Petty and T. Fraichard, "Safe motion planning in dynamic environments," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2005.

[11] T. Fraichard, "Trajectory planning in a dynamic workspace: a 'state-time' approach," *Advanced Robotics*, vol. 13, no. 1, pp. 75–94, 1999.

[12] J. van den Berg and M. Overmars, "Roadmap-based motion planning in dynamic environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 885–897, 2005.

[13] D. Nieuwenhuisen and M. Overmars, "Useful cycles in probabilistic roadmap graphs," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[14] D. Vasquez, F. Large, T. Fraichard, and C. Laugier, "High-speed autonomous navigation with motion prediction for unknown moving obstacles," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[15] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime Dynamic A*: An Anytime, Replanning Algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.