An Efficient Hybrid Planner in Changing Environments

Michael Barbehenn‡ Pang C. Chen† Seth Hutchinson‡

‡Artificial Intelligence Group
The Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
Urbana, IL 61801

†Sandia National Laboratories
Albuquerque, NM 87185

Abstract

In this paper, we present a new hybrid motion planner that is capable of exploiting previous planning episodes when confronted with new planning problems. Our approach is applicable when several (similar) problems are successively posed for the same static environment, or when the environment changes incrementally between planning episodes. At the heart of our system lie two low-level motion planners: a fast, but incomplete planner (which we call LOCAL*), and a computationally costly (possibly resolution) complete planner (which we call* GLOBAL*). When a new planning problem is presented to our planner, an efficient meta-level planner (which we call* MANAGER*), decomposes the problem into segments that are amenable to solution by* LOCAL*. This decomposition is made by exploiting a task graph, in which successful planning episodes have been recorded. In cases where the decomposition fails,* GLOBAL *is invoked. The key to our planner's success is a novel representation of solution trajectories, in which segments of collision-free paths are associated with the boundary of nearby obstacles. Thus we effectively combine the efficiency of one planner with the completeness of another to obtain a more efficient complete planner.*

1 Introduction

Robot motion planning is the problem of computing a collision-free trajectory from one robot configuration to another. Although some work has been done to cope with changing environments in which the changes are known completely in advance *e.g.* [5, 7, 6], little work has been done to cope with environments that change in unknown ways. Some local planners are able to cope with environments changing in unknown ways *e.g.* [10] but suffer from problems of local minima.

The basic idea for this paper is given in [3, 4], which describe the integration of an efficient local planner with a global planner to produce a more efficient global planner for robot manipulators. In this paper, we present a full implementation for mobile robots, and extensions.

This paper addresses three issues in robot motion planning. (1) We address the issue of solving *many* different planning problems in a given environment. Thus we want to save the results from solving one planning problem so that they can be applied to help solve similar planning problems in the future. (2) We address the issue of making use of multiple planning methods so that each method can be used to its best advantage. In particular, we present a hybrid planner that relies on a local planner for efficiency and a global planner for completeness. This results in a new, faster, global planner. We do not study the selection of one planner over another, nor how to decompose a planning problem into subproblems, each specially suited for particular planners *e.g.* [9]. (3) We address the impact of incrementally changing environments on our saved planning results.

In this paper, we examine the problem of a polygonal robot amid polygonal obstacles in the plane. We reduce this problem to that of finding a path in the three dimensional configuration space of the robot, $\mathcal{C} = \mathbf{R}^2 \times S^1$. All configurations $q \in \mathcal{C}$ for which the robot intersects some obstacle belong to the set of configuration space obstacles, denoted by $\mathcal{CB}$. For all other configurations, the robot is in free space, denoted by $\mathcal{C}_{free}$. A planning problem is specified by an initial and a goal configuration, q_{init} and q_{goal} respectively. A solution trajectory is a continuous mapping $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, such that $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$. Because obstacles are allowed to move between planning problems, $\mathcal{CB}$ and $\mathcal{C}_{free}$ change incrementally over time.

In this paper, as in [3, 4], we will restrict our attention to a single global planner and a single local planner, which we will refer to here collectively as "the workers." The local planner (LOCAL) is assumed to be extremely fast, but incomplete; the global planner (GLOBAL) is assumed to be quite slow but complete. The implica-

tion here is that LOCAL can only solve a small number of planning problems by itself. The fundamental goal of this paper is to give an effective mechanism through which LOCAL can be used to solve many different planning problems over time, in an environment in which the obstacles are not (necessarily) stationary. This mechanism is itself a planner that is responsible for coordinating and delegating planning problems to the workers. We will refer to this top-level planner as "the manager."

The manager (MANAGER) maintains an abstract *task graph* with which it keeps track of planning problems LOCAL is known to be able to solve. Each vertex $v \in V$ in the task graph $\mathcal{T}(V, E)$ represents a robot configuration q; and each edge $(v_i, v_j) \in E$ in $\mathcal{T}$ indicates that LOCAL is able to solve for the robot motion from the configuration associated with one vertex to the configuration associated with the other. Task graphs are similar to, for example, the connectivity graph of free space for an exact cell decomposition method. In that case, the configurations associated with vertices are the sample points for each cell of the cylindrical algebraic decomposition, and edges connect the vertices corresponding to adjacent empty cells [11, 1].

Given a planning problem "move from robot configuration q_{init} to configuration q_{goal}" (that LOCAL cannot solve directly) the manager searches $\mathcal{T}$ for a sequence of subproblems that, when given consecutively to LOCAL, will solve the overall planning problem. There are three phases to this process.

1. MANAGER finds a starting vertex v_i in $\mathcal{T}$ such that LOCAL can solve for the robot motion between q_{init} and v_i.
2. MANAGER finds a goal vertex v_g in $\mathcal{T}$ such that LOCAL can solve for the motion between v_g and q_{goal}.
3. MANAGER finds a path in $\mathcal{T}$ connecting v_i and v_g.

Thus MANAGER is able to use LOCAL to solve a planning problem that LOCAL is unable to solve by itself. In the event that MANAGER lacks sufficient information to enable LOCAL to solve the planning problem, MANAGER invokes GLOBAL to solve the planning problem. In this case MANAGER incorporates the resulting solution trajectory into $\mathcal{T}$ for future reference. How this is done will be discussed in Section 2.

The process, outlined in steps 1-3 above, is quite similar to planning methods that rely on a global roadmap of free space, such as a Voronoi diagram. For example, first the initial and goal configurations are "retracted" onto the roadmap, and then the roadmap is searched for a connecting path.

The task graph will change dynamically over time in two ways. (1) The solution trajectory returned by GLOBAL is decomposed by MANAGER into subproblems suitable for LOCAL. New vertices are created to represent the initial and goal configurations of the subproblems, and added to $\mathcal{T}$. New edges are created to interconnect the new vertices, and to connect the new vertices with the old. (2) As obstacles move, portions of $\mathcal{T}$ may become *invalid*. The configurations associated with vertices may no longer be in $\mathcal{C}_{free}$, or LOCAL may no longer be able to solve the planning problem represented by an edge.

The task graph is not necessarily connected, even if free space *is* connected. The task graph represents portions of trajectories in the robot's configuration space that have been computed during previous planning problems. Only those trajectories for planning problems that LOCAL is unable solve on its own are saved in $\mathcal{T}$. Not only is $\mathcal{T}$ limited to the set of planning problems that have occurred, but also by how they have been incorporated into $\mathcal{T}$. This will be discussed in more detail in the remaining sections.

In this paper we will address the underlying implementation issues that make this approach both feasible and practical. The remainder of the paper is organized as follows. In Section 2, we describe the manager in detail. Section 3 examines the nature and structure of $\mathcal{T}$. Then Section 4 evaluates several methods of saving planning results. In Section 5 we present an efficient storage mechanism which allows effective use of LOCAL to obtain an efficient and complete planner. Finally, Section 6 gives our conclusions.

2 The Manager

The manager is the primary planner responsible for solving a given planning problem. The manager solves a planning problem by coordinating the efforts of the workers. In this paper we expect MANAGER to optimize planning time, rather than solution quality (*e.g.* execution time or trajectory length). Thus we assume that it is always best to use LOCAL whenever possible.

The first thing MANAGER does with a new planning problem is see if LOCAL can solve the planning problem outright. If LOCAL succeeds, nothing more is done. In general, storing such "easy" planning problems increases the size of $\mathcal{T}$ without contributing to the completeness of the MANAGER-LOCAL team, so such solutions are not added to $\mathcal{T}$.

If LOCAL cannot solve the problem on its own, MANAGER follows the three steps described in Section 1. Note that following the three steps is not necessarily a straight-forward process, as is illustrated in Figure 1. In the figure, $\mathcal{C} = \mathbf{R}^2$, and q_{init} and q_{goal} are depicted as hollow circles. Also in the figure, the vertices in $\mathcal{T}$ are depicted by solid circles, and the edges, which represent linear trajectories, are depicted by solid line segments. Potential trajectories between q_{init} and v_i and between v_g and q_{goal} are drawn with dotted lines. If no path exists between vertices v_i and v_g in $\mathcal{T}$, then other choices for v_i and/or v_g must be made. Let V_i be the set of all vertices v_i such that LOCAL can solve for the robot motion between q_{init} and v_i; and let V_g be the set of all vertices v_g such that LOCAL can solve for the robot motion between v_g and q_{goal}. In the worst case, all possible pairings between vertices in V_i and V_g must be examined in the search for a path. Note, however, that we can eliminate from V_i and V_g all but a single member from each connected component of $\mathcal{T}$. Similarly, we can

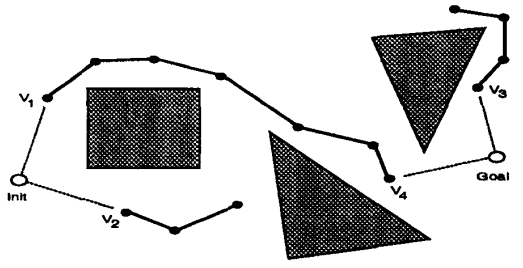

Figure 1: Multiple Initial and Goal Vertices in a Disconnected Task Graph.

compute the connected components of $\mathcal{T}$ and restrict our search for v_i and v_g to each component in turn. In Figure 1, $V_i = \{v_1, v_2\}$ and $V_g = \{v_3, v_4\}$.

The manager does not know *a priori* which obstacles will move, how they will move, or by what magnitude they will move. It is assumed that the obstacles will not move during the course of planning or execution, and that all obstacle locations are always known at planning time.

Changes in the robot's environment may result in the invalidation of some of the solutions stored in $\mathcal{T}$. If, in the course of planning, an edge or a vertex is discovered to be invalid, it is deleted from $\mathcal{T}$. Once MANAGER has obtained a path in $\mathcal{T}$, the subproblems represented by the path must be verified by LOCAL. If some vertex or edge is invalid, an alterate path must be found. By dynamically maintaining the shortest paths tree in $\mathcal{T}$, the best path is always immediately available [2].

An important part of the manager's task is to cache for future use solutions computed by GLOBAL. If GLOBAL yields a solution trajectory τ, the manager decomposes τ into subproblems suitable for LOCAL. The manager then incorporates these subproblems into $\mathcal{T}$ for future use. If, for example, GLOBAL returns piecewise linear solutions in $\mathcal{C}_{free}$, $\tau = q_1 q_2 \cdots q_n$ and LOCAL can execute any linear trajectory, then an obvious decomposition suitable for LOCAL is the sequence of configurations $q_1, q_2, \ldots q_n$. In this case, the configurations will be associated with new vertices $v_1, v_2, \ldots v_n$; and new edges will be created at least between each consecutive pair of new vertices. Additional edges might also be created between new vertices and other vertices in $\mathcal{T}$.

In general, MANAGER must verify the decomposition by having LOCAL solve the proposed subproblems. In our example, LOCAL would be asked to solve the $n-1$ problems: q_1 to q_2, q_2 to q_3, and so on, edges and vertices being created only *after* LOCAL has demonstrated its ability to solve the corresponding problem.

The basic issues that must be addressed for an efficient implementation of MANAGER are efficient storage and retrieval of previous solution trajectories τ_i found by GLOBAL, and effective selection of vertices from τ_i. To understand these issues better, we must look at the basic quantities that are being manipulated, and how they are affected by incremental changes in the robot's environment. There are three general issues that must be addressed in the context of moving obstacles: the size of V, the density of E, and the representation of configurations associated with V.

3 Task Graph Flexibility and Redundancy

Ideally, we would like $\mathcal{T}$ to represent the topology of $\mathcal{C}_{free}$ in some minimal way. So, it would be best if $\mathcal{T}$ had as few vertices as possible. Fewer vertices mean less computational overhead due to graph maintenance. Extra edges, on the other hand, provide shorter solution trajectories, in general, and add redundancy in case obstacle motion invalidates some edges.

When MANAGER incorporates a solution trajectory returned by GLOBAL, new vertices are added to $\mathcal{T}$. In order to introduce as many edges as possible, for each new vertex v added to $\mathcal{T}$, new edges need to be created connecting v to every vertex v' in $\mathcal{T}$ such that LOCAL is able to plan the motion between v and v'. In other words, MANAGER needs to have LOCAL attempt to solve $|V|$ planning problems for each new vertex, where $|V|$ is the number of vertices currently in $\mathcal{T}$.

In order to maintain as few vertices as possible, MANAGER must be conservative when it decides whether a new vertex v *needs* to be introduced into $\mathcal{T}$. Let the *neighborhood* of a vertex v be the union of $\{v\}$ with the set of vertices to which edges can be found. One criterion for deciding whether to introduce a new vertex into $\mathcal{T}$ is to compare the neighborhood of v with that of another vertex v'. If one neighborhood is a subset of the other, that vertex does not need to be in $\mathcal{T}$. Assuming MANAGER is concurrently introducing as many edges as possible, then every graph neighbor of v would need to be examined under the neighborhood subset relation. The problem with this criterion is that it depends on the current vertex set and does not reflect the true accessibility of a particular vertex with respect to $\mathcal{C}_{free}$. Under this criterion, if there were two vertices in a corridor, placed on either side of an intersection, MANAGER would not introduce a new vertex at the intersection because it does not contribute to motion within the corridor, although such a vertex might be sufficient to enable LOCAL to branch into adjoining corridors in the future.

Another criterion would restrict the neighborhood of a vertex to be contained within the current solution trajectory. In other words, if $\tau = q_1 q_2 \ldots q_n$ is the current trajectory, and the configuration associated with v is such that LOCAL can plan from some q_{i-1} to v and from v to q_{i+1}, then a vertex for q_i is not needed. The problem with this definition is that it is possible that every *single* configuration on τ is needed, but some subsequence of the configurations on τ may not be needed.

It should be noted that while a graph with a minimal number of vertices may seem computationally attractive, it is not necessarily a good idea. Besides the extra

time spent carefully incorporating new vertices into $\mathcal{T}$, another drawback is the loss of redundancy and flexibility alluded to earlier. Especially as obstacles move, and gaps between obstacles open and close, it is convenient to have extra vertices to "tap" into. Specifically, it is not the number of vertices we want to worry about, so much as it is the selection, or placement, of the vertices. We do not address vertex selection further here, edges are discussed further in Section 5.

4 Vertex Representation

This section addresses the issue of how the configurations associated with the vertices of $\mathcal{T}$ are represented. We also look at how this impacts on $\mathcal{T}$ for two different classes of solution trajectories.

If the environment is static, then a reasonable implementation of $\mathcal{T}$ has robot configurations specified in world coordinates associated with the vertices [3, 4]. This method of storage works especially well with most global planners which return solution trajectories that give good clearance from obstacles. Such trajectories are approximately equidistant from the nearest obstacles, with the exception of the trajectory endpoints which are influenced more by the initial and goal configurations than by the intervening obstacles. This storage method, in conjunction with such trajectories, provides the greatest leeway for obstacles to move without invalidating any edges or vertices in $\mathcal{T}$, as illustrated in the leftmost portion of Figure 2.

In the figure, $\mathcal{C} = \mathbf{R}^2$ so that the obstacles correspond to $\mathcal{CB}$; and LOCAL is restricted to linear trajectories in $\mathcal{C}_{free}$ so that the edge depicted corresponds geometrically to the implicit trajectory. In the figure, both obstacles can move simultaneously towards the edge the full distance separating them ($d/2$ where d is the distance separating the two obstacles). The total displacement allowed is therefore $d = d/2 + d/2$. If, on the other hand, the solution trajectory is close to $\mathcal{CB}$, then some obstacles have great freedom of motion, while others have little. This is illustrated in the right half of the left-hand side of the figure. This is a very brittle, and undesirable, situation in that such edges are at high risk of being deleted. In the figure, the lower obstacle cannot move towards the edge without invalidating it, while the upper obstacle can move the full distance d towards the edge. Again, in this case the total displacement allowed is d.

If the environment is dynamic, then not only may obstacles move and so block the trajectory represented by edges, but the configurations associated with vertices may no longer be in $\mathcal{C}_{free}$. This is especially problematic for portions of $\mathcal{T}$ that represent trajectories near $\mathcal{CB}$. An alternative approach is recommended in [3, 4], in which a configuration associated with a vertex is represented relative to the nearest obstacle at the time MANAGER incorporated the vertex into $\mathcal{T}$. Thus if an obstacle moves, the vertices "attached" to that obstacle move with it, maintaining their clearance.

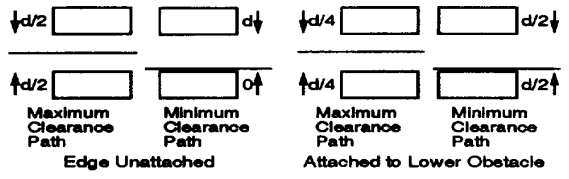

Figure 2: Maximum Simultaneous Displacement of Two Obstacles Near an Edge in $\mathcal{T}$.

Note that this "object attachment" sometimes reduces the mobility of adjacent obstacles in the sense that they are now more likely to invalidate an edge than before. This is illustrated in the right half of Figure 2. With object attachment and an edge representing a trajectory that is equidistant from two obstacles, the total displacement is reduced to $d/2$. This results from the lower obstacle "pushing" the edge upwards as it moves. However, for an edge representing a trajectory near $\mathcal{CB}$, that obstacle is now free to move the full distance towards the other obstacle, and vice versa. In contrast, the static edges allowed no motion for this obstacle in the direction of the edge. In this case the total displacement allowed is d. This is illustrated on the rightmost portion of the figure.

In cluttered environments, the distinction between solution trajectories that skirt obstacle boundaries and those that maximize clearance becomes blurred. In cluttered environments, most solution trajectories are close to obstacles. Once a vertex near an obstacle is attached to that obstacle, it will remain near that obstacle. Thus both classes of solution trajectories, those with minimal clearance, and those with maximal clearance, converge to similar representations.

Another effect of object attachment is that as obstacles translate and rotate, they might "push," "pull," or "swing" vertices into other obstacles; and they might "stretch" edges across obstacles. This situation is depicted in Figures 3 and 4. In the figures, $\mathcal{C} = \mathbf{R}^2$, and as the shaded obstacle moves, it causes the attached vertices, also shown shaded, to move with it. The loss of vertices cannot be avoided to a large extent: either the topology of $\mathcal{C}_{free}$ has changed, or the vertices are too close to other obstacles. The loss of edges can, however, be avoided to a large extent. The majority of edges are lost due to the *interleaving* of obstacles along the path in $\mathcal{T}$, as explained below.

As the solution trajectory winds past obstacles, the configurations along the trajectory are associated with vertices attached to different obstacles. This interleaving of obstacle associations can lead to either an unnecessarily high loss of edges when one of these obstacles moves, or to a distorted and inefficient solution trajectory. Two examples of edge loss are depicted in Figures 3 and 4. Two examples of path distortion are depicted in Figures 5 and 6. As one obstacle moves away from another obstacle, alternate segments of the represented

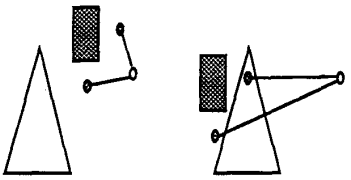

Figure 3: Loss of Edges and Vertices due to Obstacle Translation.

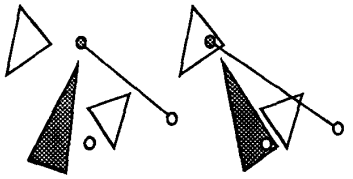

Figure 4: Loss of Edges and Vertices due to Obstacle Rotation.

trajectory between the two obstacles are pulled in opposite directions, deforming the trajectory into a zig-zag pattern. Interleaving is exaggerated when GLOBAL returns solutions that are equidistant to multiple obstacles (yielding maximal clearance), which is common in many global planners [8].

5 Decoupling Trajectory Segments

One solution to the problem of interleaving is to decouple motion around obstacles from motion between obstacles. One way to achieve this decoupling is to introduce two tiers of graphs: a high-level plan-graph $\mathcal{P}$, and low-level obstacle graphs $\mathcal{G}^i$. As GLOBAL produces solution trajectories near obstacle i, vertices attached to the obstacle are created and added to $\mathcal{G}^i$. The first time this occurs, $\mathcal{G}^i$ is created, and added as a vertex to $\mathcal{P}$. Edges are added to $\mathcal{G}^i$ to represent trajectories around the boundary of obstacle i; and edges are added to $\mathcal{P}$ to represent trajectories between obstacles.

The task graph relates to $\mathcal{P}$ and $\mathcal{G}^i$ the following way. The vertices of $\mathcal{T}$ are partitioned into sets, one set per $\mathcal{G}^i$. For every edge in $\mathcal{T}$ that connected vertices attached to the same obstacle i, there might be a corresponding edge in $\mathcal{G}^i$. As we will see below, an improved storage mechanism allows for a more dense, more systematic,

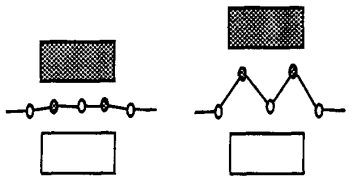

Figure 5: Zig-zag Path Produced as a result of Obstacle Translation.

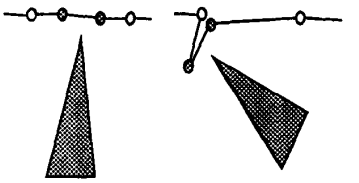

Figure 6: Zig-zag Path Produced as a result of Obstacle Rotation.

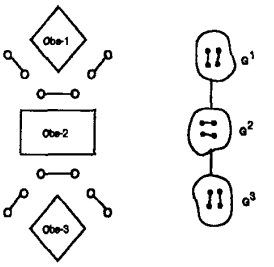

Figure 7: Ambiguous Plan Graph.

and more effective set of edges. Associated with the edges of $\mathcal{P}$ is some extraction from the edges of $\mathcal{T}$ that connected vertices attached to one obstacle with those of another.

This decoupling prevents interleaving as identified above. There are no more edges connecting vertices associated with one obstacle with vertices associated with another obstacle. Instead, all such edges are replaced with a single, abstract edge between obstacle graphs. Since there are fewer edges corresponding to trajectories between obstacles there is less maintenance when obstacles move.

This means of decoupling eliminates interleaving, but it introduces the problem of disconnected trajectory segments associated with an obstacle. If, for example, the robot has approached an obstacle from opposite sides, there will be vertices associated with that obstacle that cannot be joined by LOCAL. This is illustrated in Figure 7. The manager must therefore be able to reason about non-simple paths in $\mathcal{P}$. One way to do this is to annotate each instance of a vertex on the path in $\mathcal{P}$ with specific obstacle graph entry and exit points *e.g.* [12]. The manager must also be able to identify false connections as illustrated in Figure 7. A false connection is a path in $\mathcal{P}$ that is not supported by the underlying obstacle graphs. In the figure, there is a plan graph edge connecting $\mathcal{G}^1$ with $\mathcal{G}^2$, and an edge connecting $\mathcal{G}^2$ with $\mathcal{G}^3$, however $\mathcal{G}^2$ is disconnected and the plan graph path cannot be realized.

Instead, our manager does not incorporate trajectories, or even segments of trajectories, verbatim. The obstacle graph vertices are maintained sorted by the polar coordinates (r, θ) of the origin of robot coordinate frame relative to the local obstacle coordinate frame, primarily

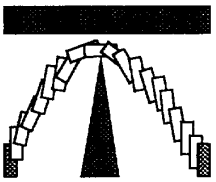

Figure 8: Sample Problem and Environment.

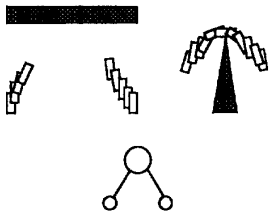

Figure 9: Resulting Obstacle Graphs and Corresponding Plan Graph.

by the angle θ and secondarily by distance r. These values are invariant under obstacle motion, as is the relative orientation of the robot with respect to the obstacle to which it is attached. When a new vertex is introduced to an obstacle graph, LOCAL is used to attempt to fully interconnect the vertex with its nearest neighbors, both radially and, within the same orientation, by distance.

The plan graph vertices no longer correspond to obstacle graphs in entirety, but rather to the connected components of the obstacle graphs. In this way we eliminate the need to reason about non-simple paths and false connections in $\mathcal{P}$. All critical relationships are made explicit. As new vertices are added to obstacle graphs and interconnections added, connected components, and hence plan graph vertices, are merged. Similarly, as obstacle graph vertices are discovered, in the course of planning, to be invalid, they are deleted and the connected component subdivided as necessary.

An example of actual obstacle graphs and the corresponding plan graph our manager creates is given in Figure 9 for the planning problem, shown shaded, in Figure 8. In this example, the environment consists of a tall triangle and a wide rectangle ($\mathcal{C} = \mathbf{R}^2 \times S^1$); and LOCAL is restricted to linear trajectories in $\mathcal{C}_{free}$. The manager is asked to solve for the motion of the small rectangle from the left side of the triangle to the right side. In this example, LOCAL cannot solve the problem by itself, and there is no plan graph yet. The global planner returns the solution trajectory illustrated in Figure 8 which is piecewise linear in $\mathcal{C}_{free}$. The configurations along this trajectory are associated with the obstacles, and an abstract plan graph is created. These are illustrated in Figure 9.

6 Conclusions

In this paper we presented a fully implemented hybrid motion planner that exploits the completeness of a global planner and the speed of a local planner to form a fast global planner that is robust under changing environments. The key to the successful implementation of the planner lies in the representation of solution trajectories provided by the global planner for future use by the local planner. For our representation we chose to maintain a graph of connected obstacle graph components. This allows us to navigate successfully around obstacles, even after they have moved.

Acknowledgements: This research was supported by the U.S. Department of Energy and Sandia National Laboratories under contract DE-AC04-76DP00789 and the National Science Foundation under grant number NSF-IRI-9110270. This paper benefited from the comments of Y. Hwang and P. Xavier, and from the encouragement of D. Strip.

References

[1] D. S. Arnon. Geometric reasoning with logic and algebra. *Artificial Intelligence*, 37(1-3):37–60, Dec 1988.

[2] M. Barbehenn and S. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. In *IEEE Int'l Conf. on Robotics and Automation*, vol 1, pages 566–571, 1993.

[3] P. C. Chen. Improving path planning with learning. In *Proc. Machine Learning Conference*, pages 55–61, 1992.

[4] P. C. Chen. Adaptive path planning in changing environments. Report SAND92-2744, Sandia National Laboratories, 1993.

[5] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1419–1424, 1986.

[6] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *IEEE Int'l Conf. on Robotics and Automation*, vol 1, pages 560–565, 1993.

[7] K. Fujimura. Motion planning using transient pixel representation. In *IEEE Int'l Conf. on Robotics and Automation*, vol 2, pages 34–39, 1993.

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

[9] S. Pandya and S. A. Hutchinson. A case-based approach to robot motion planning. In *Proc. of the IEEE Int'l Conf. on SMC*, pages 492–497, 1992.

[10] S. Ratering and M. Gini. Robot navigation in a known environment with unknown moving obstacles. In *IEEE Int'l Conf. on Robotics and Automation*, vol 3, pages 25–30, 1993.

[11] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. In J. T. Schwartz, M. Sharir, and J. Hopcroft, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 51–96. Ablex, Norwood, NJ, 1987.

[12] D. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Trans. on Robotics and Automation*, 7(1):9–20, February 1991.