Practical Path Planning among Movable Obstacles*

Pang C. Chen Yong K. Hwang
Sandia National Laboratories
Albuquerque, NM 87185

September 5, 1990

Abstract

Path planning among movable obstacles is a practical problem that is in need of a solution. In this paper, we present an efficient heuristic algorithm that uses a generate-and-test paradigm: a "good" candidate path is hypothesized by a gloɒal planner and subsequently verified by a local planner. In the process of formalizing the problem, we also present a technique for modeling object interactions through contact. Our algorithm has been tested on a variety of examples, and was able to generate solutions within 10 seconds.

1 Introduction

Path planning for mobile robots is a key component in robotics, and has received much attention. Various approaches have been used to plan short and safe paths. Most of the previous work, however, has considered environments with stationary obstacles, obstacles moving along prescribed trajectories, or the control of multiple robots. In the movable obstacle problem, the robot is allowed to move objects in the environments to clear a path for itself. This problem is very important and practical since mobile robots often encounter situations in which objects have to be moved out of the robot's way. Unfortunately, however, the movable-obstacle problem has been shown to have high complexity [Wilf88], and an exact algorithm is not expected to run within practical limits. On the other hand, allowing the robot to move objects creates more solutions than in a fixed environment. In this respect, practical path planning among movable obstacles should be easier than among fixed

*This work has been performed at Sandia National Laboratories and supported by the U.S. Department of Energy under Contract DE-AC04-76DP00789.

obstacles. We demonstrate this point by presenting a fast, heuristic algorithm that solves the movable-obstacle problem in a variety of situations.

Several approaches have been used to plan paths for mobile robots. The visibility graph is used when the shortest path is desired [LoWe79, RaIS88, FuSa90]. The paths in the visibility graph touch the corners of obstacles to minimize the path lengths. Retraction approaches use the Voronoi diagram or the medial axes of the free space [OdYa82, HwAh89] to generate paths that stay as far away from obstacles as possible. Cell decomposition techniques build obstacle-free cells of the free space, and find paths from the graph representing the adjacency information of the cells. The cell shapes can either be uniform such as in quadtree [KaDa86, NoNA89], or dependent upon the boundaries of obstacles [Nguy84, RuWo87, SiWa87]. Grid representations of environments are used for two cases. When the environment consists of regions with different traveling costs, a search for the minimum-cost path through the grid can be conducted [GaMe86, AlRo90]. When a safety margin is desired between the robot and the obstacles, each point on the grid is assigned a cost based on an artificial potential that increases as the robot approaches obstacles [KrTh85, Warr89, HaBC90]. The minimum-cost path found then represents the shortest path with the desired amount of safety margin.

Other variations of the path planning problem for mobile robots include the moving obstacle problem [ReSh85, KaZu88, FuSa90], the multi-robot problem [ScSh83, Buck89, LiKN89, PaCa90], and the unknown environment problem [LuSt87, SaVi90]. Both exact and heuristic algorithms have been developed to solve these problems. There appears to be only one paper that examines the movable obstacle problem. Wilfong [Wilf88] has shown that the movable-obstacle problem is PSPACE-hard when the final positions of obstacles are specified, and otherwise, NP-hard. He has presented an $O(n^3 \log^2 n)$ algorithm for the case of one movable obstacle. His algorithm computes a planar partition of the possible positions of the movable object so that within a particular partition, the free space has essentially the same connectivity properties.

This paper presents an effective approach that yields a solution in seconds to an otherwise intractable problem. It is essentially grid based, and makes judicious use of a generalized distance function to move the robot and the obstacles. This paper is organized as follows: The movable obstacle problem is formulated in Section 2, and our algorithm is presented in Section 3. Its performance is illustrated and evaluated in Sections 4 and 5.

2 Problem Formulation

Let R be a mobile robot in a workspace filled with movable objects, each with a positive weight. Let a state be a description of the workspace with the location and orientation of each object and the robot totally specified. Given an initial state with the robot at position s, and a goal position for the robot at t, our problem is to find a short path for R that also minimizes the work required in moving obstacles out of the path. More specifically, if we let $\Gamma(R)$ be a robot path from s to t, and $\Gamma(B_i)$ be a path for each object B_i with weight w_i, then we are interested in finding a feasible path $\Gamma(R)$, while minimizing the length of $\Gamma(R)$ plus the weighted length of $\Gamma(B_i)$ for each B_i.

We assume that the robot has the capability to push any object in any direction, once it is in contact with that object. (In our implementation, the robot is assumed to have an invisible arm that can move the touched object in any direction.) While an object is being pushed by the robot, it may push other objects that it contacts. To model the contact interactions between objects, we will allow an object to penetrate another slightly before the penetrated one can move in a direction that minimizes the overlap.

To facilitate the discussion and implementation of our algorithm, we assume a circular robot in a two-dimensional workspace in which each object is represented by a union of convex polygons.

3 Algorithm

Our algorithm employs a generate-and-test strategy: We first generate a "good" sequence of subgoals with a "global planner", $\mathcal{G}$, and then test the reachability of each subgoal with a "local move" module, $\mathcal{L}$. If a subgoal cannot be reached, then the next best sequence of subgoals will be considered until either no sequence is available or a feasible path is found.

In our algorithm, we use the generalized distance, d, that extends the Euclidean distance between objects. The concept of d is described in [Buck85], and is used in [PaMF89] to plan paths for manipulators. When two objects A and B do not intersect, $d(A, B)$ corresponds to the minimum separation distance; otherwise, $d(A, B)$ is negative, and its magnitude corresponds to the minimum distance A has to be translated to be separated from B. We use d in this paper in three important ways. First, it is used in the global planning to estimate potential feasibility of a subgoal. In the local move module, it is used to find a direction to move objects away from the robot's predetermined path. The generalized distance is also used to model object interactions during contact.

3.1 Global Planning

The global planner $\mathcal{G}$ finds a good sequence of subgoals by searching through a dynamic graph of subgoals. The initial structure of this graph G with nodes $U \cup V$ and edges E is constructed as follows: First, we subdivide the workspace into square cells with a predetermined size, and set $U = \{s\}$, $V = \{t\}$, where s and t are the start and goal positions of the robot, respectively. Then, we randomly sample each cell for a subgoal position v to be placed in V. Our heuristic is to select a v that minimizes the node cost

$$f(v) = -\sum_i \min(0, w_i d(B_i, R)),$$

which represents a lower bound on the work required to move all objects away from the robot at v. (Notice that $d(B_i, R)$, when negative, represents the necessary distance that B_i has to travel to avoid a collision with R.) Finally, for each pair of positions u, v belonging to the same or adjacent cells, we construct an edge $e = (u, v)$ with edge cost $g(e)$ equal to the Euclidean distance from u to v.

With G initialized, we can now plan a good sequence of subgoals by finding a short path in G that connects s to t. To find this path efficiently, we will not consider other ways of reaching a subgoal once it is declared reachable. Thus, for each reachable subgoal, we will have just one associated state that describes the positions of all objects when the robot reaches the subgoal. To avoid reentering a subgoal, we will let U keep track of the set of nodes known to be reachable from s, and let V be its complement in G. Since no subgoal can be entered twice, it is only necessary for us consider simple paths in G. Hence, we can apply Dijkstra's algorithm on G with node cost f and edge cost g to map out a tree T of G that gives a shortest path from s to every node in G. By tracing the path in T from t back to a node in U, we can then find our best sequence of subgoals to be tested.

To verify the feasibility of an untested sequence in the form

$$\rho = (v_1, v_2, \ldots, v_k = t),$$

we will begin with the state associated with v_1, and invoke the local move module to move the robot to v_2. If v_2 is reachable, then we will swap v_2 from V to U, associate the state thus reached with v_2, and iteratively verify the rest of the path until we reach t. If the verification process does indeed end with t reached, then obviously a feasible path from s to t can be retraced. However, if there is some edge (v_j, v_{j+1}) not connectable using the local move module, then we will penalize the edge by deleting it from G, and proceed to generate the next candidate sequence from the new G.

To generate the next candidate sequence efficiently, we can take advantage of the fact that only one edge (u, v) in T is deleted from the previous G, which means that only the shortest paths to the descendents of v in T are affected. Thus, to update T, all we have to do is to reapply Dijkstra's algorithm on the subgraph G' induced by the descendents of the unreachable node v, using the neighboring nodes of G' in G as the new multiple source. The best sequence after updating T can then again be obtained by tracing the path in T from t back to the first node in U.

The complexity of our algorithm is largely determined by n, the number nodes in G, m, the number of times that $\mathcal{G}$ calls $\mathcal{L}$, and l, the maximum number of steps required to execute $\mathcal{L}$. Since every node by construction has only a fixed number of neighbors, the number of edges in G is only $O(n)$, which incidentally implies that $m = O(n)$. Thus, with a standard min-heap implementation [AhHU74], a shortest path can be generated within $O(n \log n)$ steps. Therefore, our algorithm will take at most $O(mn \log n)$ steps in generating paths, and $O(mnl)$ steps in verifying paths, giving a total time complexity of $O(mn(\log n + l))$.

The low complexity of this algorithm is due to our crucial design decision of not considering other ways of reaching a subgoal once it is declared reachable. Of course, it is possible to design a more "complete" algorithm by considering more than one state per subgoal. The amount of extra work required, however, will be extremely prohibitive as the combinatorial explosion takes its toll. Fortunately, our primary goal is not to design a "complete" algorithm, but merely a fast module that will yield a solution for many practical problems. For problems that are too difficult for our algorithm to handle, a higher level task planner should take the responsibility of breaking down the task into easier subtasks. Therefore, we believe that our algorithm is "adequately complete" for practical purposes.

3.2 Local Planning

The objective of the local move module, $\mathcal{L}$, is to move the robot from one subgoal to another. The module should implement a fast and simple algorithm that is likely to succeed in finding a collision-free path when the subgoals are close to each other. The global planner should give close enough subgoals so that $\mathcal{L}$ has a high probability of success. The probability of success is determined by the complexity of $\mathcal{L}$. The more powerful $\mathcal{L}$ is, the more computation time it will require. Since low computation time is one of our main objectives, $\mathcal{L}$ is designed to be just powerful enough to move the robot from a node in a cell to that in an adjacent cell.

In our implementation, $\mathcal{L}$ consists of two components: *slide* and *shove_aside*. The slide module tries to move the robot toward the desired subgoal without moving objects. If the robot cannot get any closer to the goal with *slide*, then *shove_aside* is used to move the obstructing objects away from the robot's path. Since moving an object might cause the object to collide with other objects, *shove_aside* is called recursively to move subsequently colliding objects. The local move module calls *slide* again after shoving aside the objects on the robot's path. Iterative application of *slide* and *shove_aside* is performed until either the goal is reached, or obstructing objects cannot be shoved aside as described later.

The slide module uses a hill-climbing method to approach the goal while avoiding objects. When the distance to the goal cannot be decreased due to the blocking objects, a fictitious object W is created. W is a rectangular object that encloses the area swept by the robot moving from the current position to the goal in a straight line. The objects intersecting W are identified and passed to *shove_aside*.

The shove-aside module makes judicious use of the generalized distance. The generalized distance provides a fast and powerful way to move objects out of the way. Objects are shoved aside as follows: The generalized distance between W and each object is computed, and those with non-positive distances are marked as objects to be shoved. A hill-climbing method is used to move each object away from W. The

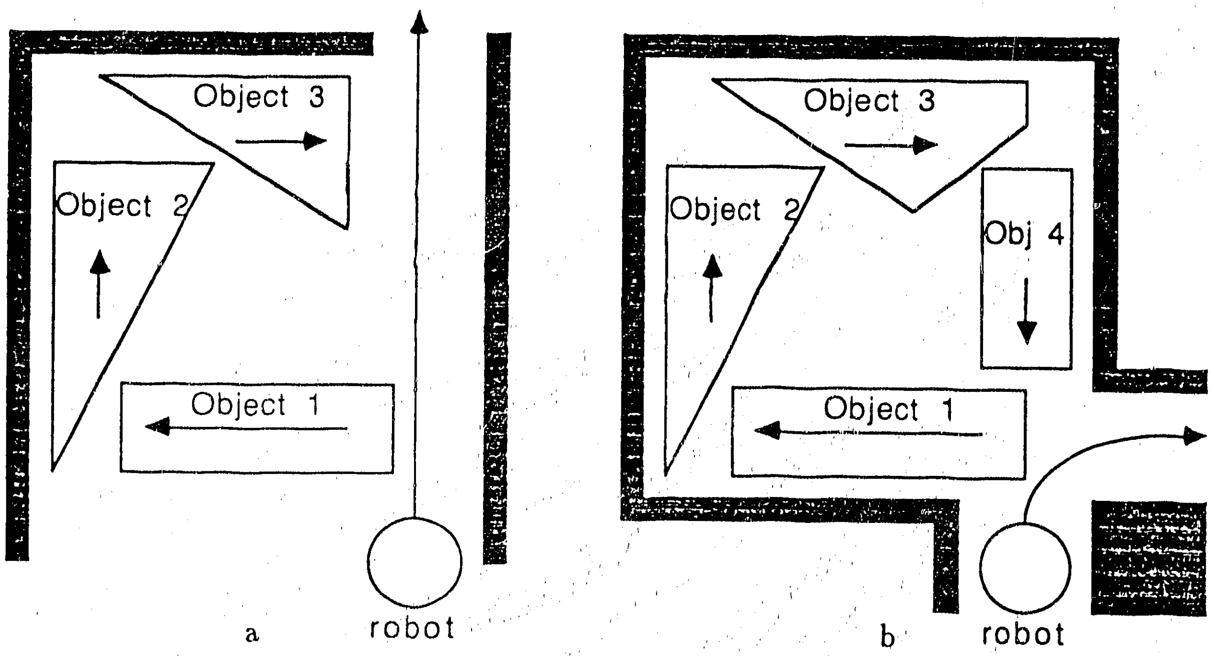

Figure 1: A case where the local move module fails.

configuration of each colliding object B_i is changed incrementally in a direction that maximizes $d(B_i, W)$ until either B_i separates from W, or collides with another object besides W. In the latter case, *shove_aside* is recursively called with W replaced by B_i. This recursive nature is in fact the reason for creating W as a fictitious object.

It may not be possible to shove obstructing objects off the robot's path completely when the accumulated weights of the objects currently being moved exceeds the robot's pushing capability. (Notice that this condition includes the encounter with immovable objects.) If this situation occurs, then the robot would use *slide* to advance toward the goal through the partially cleared path until it cannot get any closer to the goal. If the robot is able to advance more than a preset number of steps, then *shove_aside* is invoked again. Otherwise, it is assumed that the objects cannot be cleared away from the robot's path, so the local move module reports a failure to the global planner. It may happen that during the shove-aside recursion some object is pushed back into the robot's way, as shown in Figure 1a. When the robot shoves Object 1 aside, Object 3 is pushed back onto the robot's path. The robot will advance toward the goal and shove Object 3 aside. In Figure 1b, however, shoving Object 1 aside will cause Object 4 to block the robot's path, and vice versa. Such a situation can be resolved by using a more elaborate algorithm, but we chose not to include it in $\mathcal{L}$ to reduce computation time. We shift the responsibility of finding a solution in such cases to either $\mathcal{G}$, or a higher level task planner.

3.3 Alternative Approach

We have considered many other approaches, and one based on the Voronoi diagram seems promising enough to be mentioned. First, a Voronoi diagram is constructed by assuming that all objects are immovable. The best candidate path in the diagram is then selected and subsequently tested by $\mathcal{L}$. If this fails, then the next best path is selected and tested. If there are no more paths to be tested, then the lightest set of objects is deleted and a new Voronoi diagram is constructed. The process of generating candidate paths and testing with $\mathcal{L}$ is continued with the new Voronoi diagram. When testing with $\mathcal{L}$, the deleted objects are, of course, put back in the world space. If no solution is found in the current diagram, then the search process continues in the next Voronoi diagram, which is constructed by deleting the next lightest set of objects. This process is repeated until either a solution is found, or there are no more paths to be tested in the final Voronoi diagram containing only immovable objects. The advantage of this approach is that it does not examine similar paths, because the edges of the Voronoi diagram are topologically distinct. However, this approach has its disadvantage depending on how we define the Voronoi diagram. If we were to define it as the points equi-distant to two or more edges of objects, then the Voronoi diagram is very sensitive to the variations in object shapes, and in fact is ill-defined for curved objects. On the other hand, if we were to define it as the points equi-distant to two or more objects, then it would not contain edges in the space surrounded by a single concave object. Hence, if the robot were in such a space, then it would not have a path to follow. We are currently investigating ways to circumvent this problem.

4 Examples

Our algorithm has been tested on several examples, and shown to be effective. In our figures, the robot is drawn as a disk, and polygonal objects are shaded according to their weights. The darker the objects, the heavier they are. Black objects are immovable. In the first frame of each example, the start and goal positions are shown along with the sequence of subgoals generated by the global planner. The rest of the frames show the state of the objects before and after the local move module calls the shove-aside module. The frames are ordered from left to right and from top to bottom.

The example in Figure 2 has three immovable and two movable objects. The path through the left corridor is shorter, but the robot has to push the heavier rectangular block. The robot elects to travel through the right corridor, pushing the lighter triangular object. The robot in Figure 3 maneuvers through a tight space between the blocks. The S-shaped path is selected to minimize the movements of the two heavier L-shaped objects. In Figure 4, all the objects have the same weights. The global planner selects the path between the square and the L-shaped object rather than that between the square and the triangle. This is due to the higher cost at the narrow passage along the latter path. The shove-aside module is called twice in this example. The paths in these examples were each found within 10 seconds on a 17 MIPS SUN Sparc workstation. The example in Figure 5 shows a case where our

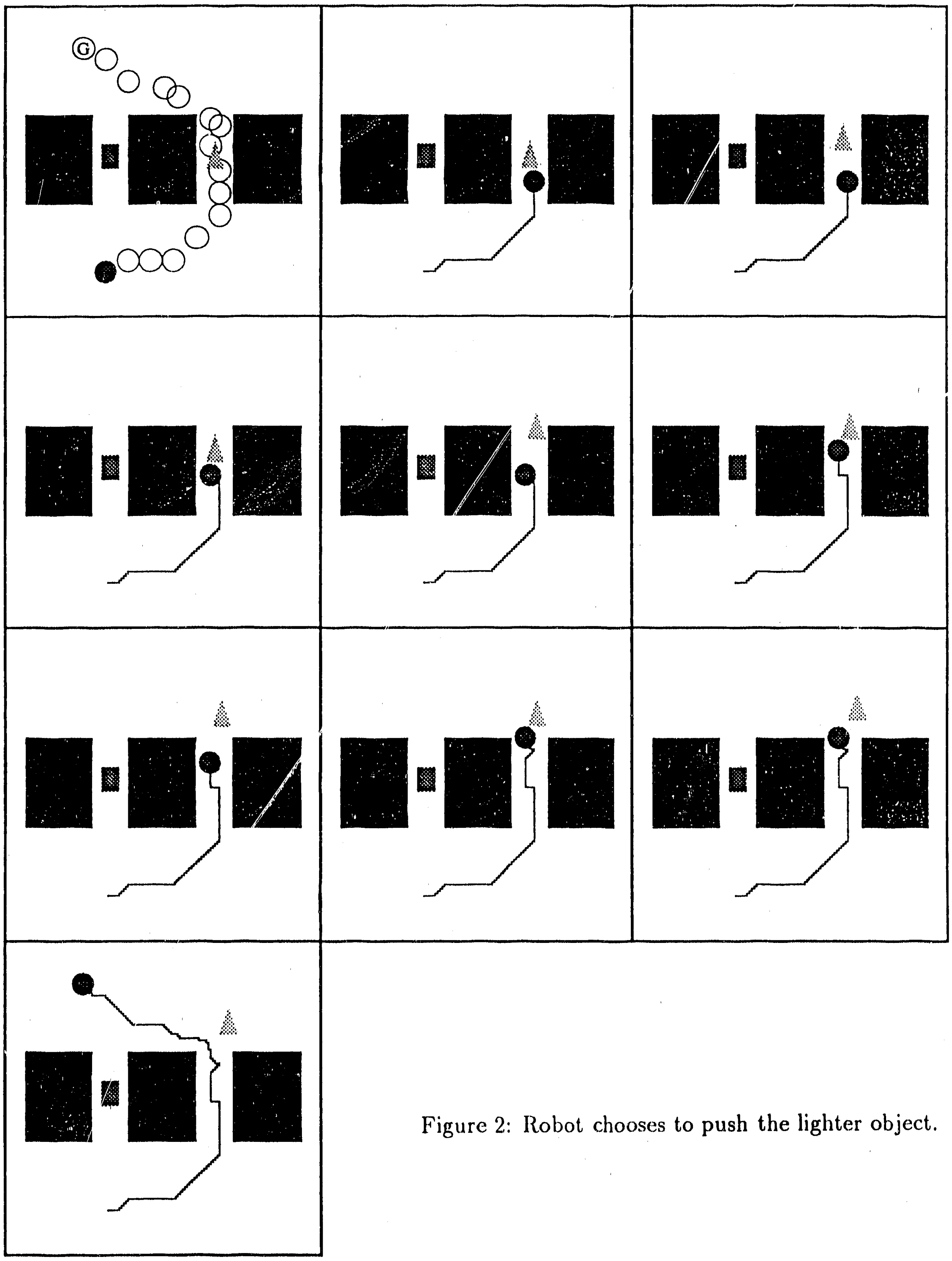

Figure 2: Robot chooses to push the lighter object.

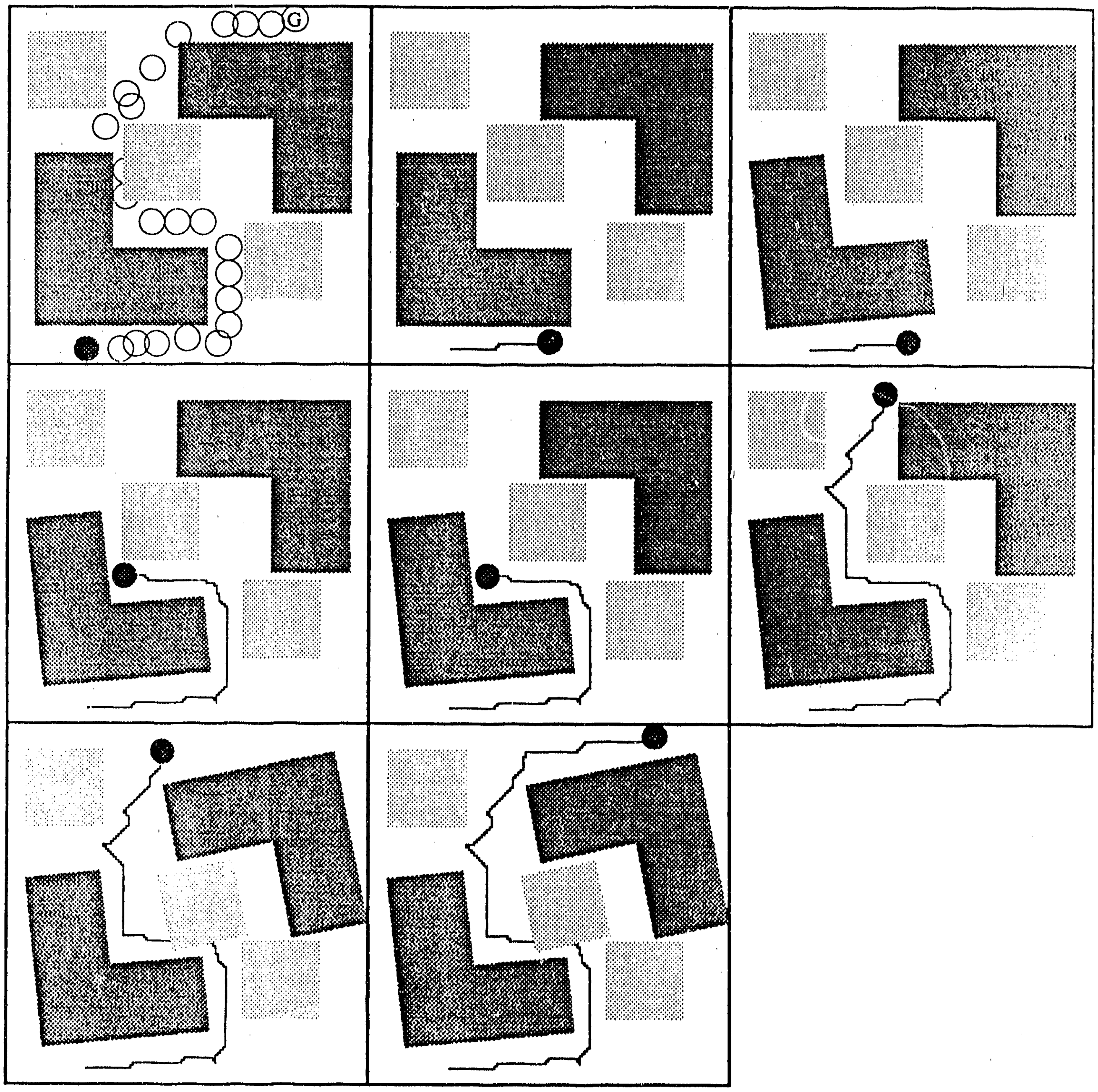

Figure 3: A generic warehouse example.

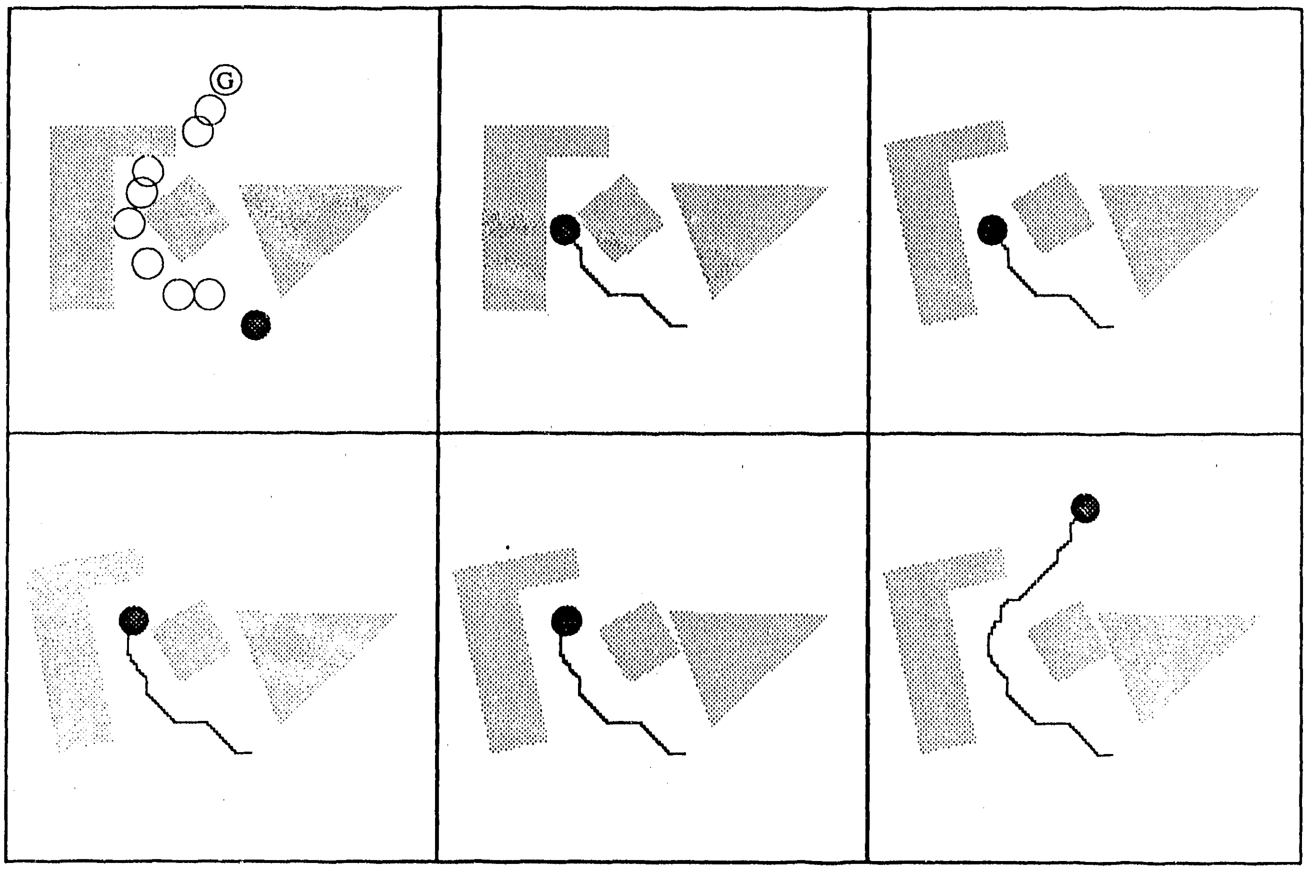

Figure 4: Robot chooses the wider path.

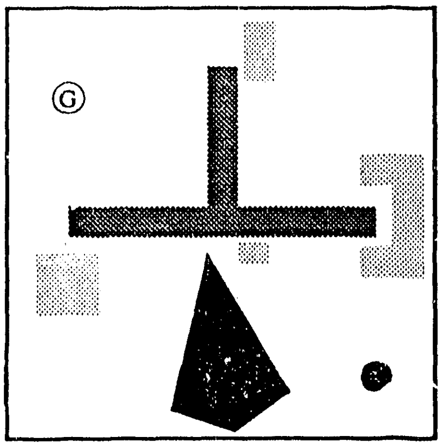

Figure 5: A difficult example.

algorithm fails to find a solution. The difficulty of this problem is approaching that of a puzzle. It requires a higher level of planning to move the U-shaped object enclosing the right tip of the T-shaped object.

5 Conclusion

Although the movable obstacle problem is theoretically difficult, it is nevertheless a realistic problem. In practical situations where objects are not arranged in a puzzle-like fashion, the problem of moving a robot to a desired position is actually easier if the robot is empowered to move objects off its path. In this paper, we have presented a computationally inexpensive, heuristic algorithm that solves the problem in many practical situations. To our knowledge, it is the first implemented algorithm for the movable obstacle problem. The effectiveness of our algorithm has been demonstrated with a number of examples. The problems for which our algorithm fails either require higher level knowledge to solve, or they are badly chosen tasks by the task-level planner. Our planner is one component of a complete robot system, and the task-level planner should share the responsibility of completing a task by providing achievable subtasks to our planner. Our algorithm can be extended to three dimensions with little modification.

References

[AhHU74] Aho, A., Hopcroft, J., and Ullman, J. *The Design and Analysis of Computer Algorithms,* Addison-Wesley, 1974.

[AlRo90] Alexander, R. and Rowe, N., "Path planning by optimal-path-map construction for homogeneous-cost two-dimensional regions," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 1924-1929, 1990.

[Buck85] Buckley, C. E., "The application of continuum methods to path planning," Ph.D. dissertation, Stanford University, 1985.

[Buck89] Buckley, S. J., "Fast motion planning for multiple moving robots," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 322-326, 1989.

[FuSa90] Fujimura, K. and Samet, H. "Motion planning in a dynamic domain," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 324-330, 1990.

[GaMe86] Gaw, D. and Meystel, A., "Minimum-time navigation of an unmanned mobile robot in a 2-1/2D world with obstacles," *Proceedings of IEEE International Conference on Robotics and Automation,* pp. 1670-1677, 1986.

[HaBC90] Hague, T., Brady, M. and Cameron, S., "Using moments to plan paths for the Oxford AGV," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 210-215, 1990.

[HwAh89] Hwang, Y. K. and Ahuja, N., "Robot path planning using a potential field representation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 569-575, San Diego, California, June 1989.

[KaDa86] Kambhampati, S. and Davis, L. S., "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 3, pp. 135-145, 1986.

[KaZu88] Kant, A. and Zucker, S., "Planning collision-free trajectories in time-varying environments: A two-level hierarchy," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1644-1649, 1988.

[KrTh85] Krogh, B. H. and Thorpe, C. E., "Intergrated path planning and dynamic steering control for autonomous vehicles," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1664-1669, 1986.

[LiKN89] Liu, Y. H., Kuroda, S., Naniwa, T., Noborio, H. and Arimoto, S., "A practical algorithm for planning collision-free coordinated motion of multiple mobile robots," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1427-1432, 1989.

[LoWe79] Lozano-Peŕez, T. and Wesley, M. A., "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560-570, October 1979.

[LuSt87] Lumelsky, V. J. and Stepanov, A. A., "Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, Springer-Verlag, 1987

[Nguy84] Nguyen, V. C., "The findpath problem in the plane," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI Memo 760, 1984.

[NoNA89] Noborio, H., Naniwa, T. and Arimoto, S., "A feasible motion planning algorithm for a mobile robot on a quadtree representation," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 327-332, 1989.

[OdYa82] C. O'Duńlaing and Yap, C. K., "A retraction method for planning the motion of of a disc," *Journal of Algorithms*, vol. 6, pp. 104-111, 1982.

[PaCa90] Parsons, D. and Canny, J., "A motion planner for multiple mobile robots," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 8-13, 1990.

[PaMF89] Paden, B., Mees, A. and Fisher, M., "Path Planning Using a Jacobian-Based Freespace Generation Algorithm," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1732-1737, 1989.

[RuWo87] Rueb, K. D. and Wong, A. K. C., "Structuring free space as a hypergraph for roving robot path planning and navigation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 2, pp. 263-273, 1987.

[RaIS88] Rao, N., Iyengar S. and deSaussure, G., "The visit problem: visibility graph-based solution," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1650-1655, 1988.

[ReSh85] Reif, J. and Sharir, M., "Motion planning in the presence of moving obstacles," *Proceedings of 25th IEEE Foundations of Computer Science*, pp. 144-154, 1985.

[SaVi90] Sankaranarayanan, A. and Vidyasagar, M., "A new path planning algorithm for moving a point object amidst unknown obstacles in a plane," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1930-1936, 1990.

[ScSh83] Schwartz, J. T. and Sharir, M.' "On the piano movers' problem: III. Coordinating the motion of several independent bodies amidst polygonal barriers," *International Journal of Robotics Research*, vol. 2, no. 3, pp. 46-75, 1983.

[SiWa87] Singh, S. and Wagh, M. D., "Robot path planning using intersecting convex shapes," *IEEE Journal of Robotics and Automation*, vol RA-3, no. 2, pp. 101-108, April 1987.

[Warr89] Warren, C. W., "Global path planning using artificial potential fields," *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 316-321, 1989.

[Wilf88] Wilfong, G., "Motion planning in the presence of movable obstacles," *Proceedings of 4th Annual Symposium on Computational Geometry*, pp. 279-288, June 6-8, 1988, Urbana-Champaign, Illinois.