Toward an Exact Incremental Geometric Robot Motion Planner*

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

Abstract

In this paper we introduce a new class of geometric robot motion planning problems that we call incremental problems. We also introduce the concept of incremental algorithms to solve this class of problems. As an example, we describe an incremental critical curve based exact cell decomposition algorithm for a line segment robot moving freely amidst polygonal obstacles. In the example, after computing an initial representation of the robot's free space, the algorithm maintains the representation as obstacles are moved between planning problems. The cost to maintain the representation is expected to be small relative to the cost of its initial construction.

1 Introduction

A geometric robot motion planner computes a collision-free trajectory between two given robot configurations. If the environment is not static, then it can change either during the planning process, during plan execution, or between planning problems. Current research in geometric robot motion planning with moving obstacles, or with movable objects, is either heuristic (*e.g.* [13, 18, 9, 25, 11, 15]), does not scale to realistic problems (*e.g.* [31, 35, 12]), or is only probabilistically complete (*e.g.* [7, 6]). In this paper, we discuss a solution to a new class of geometric robot motion planning problems that promises to extend to the more general problems of moving and movable obstacles in a way that scales and does not rely on heuristics.

We call the new class of problems *incremental problems.* A problem is incremental if its input undergoes incremental change over time. An example of incremental change is a perturbation in the position of some obstacles in the environment between planning problems. Such problems arise frequently in robotics, but to date, there has been little research in this area. Examples of such changes occur when designing the layout of a factory floor, or reconfiguring a floor plan given new equipment.

A primary attribute of an incremental problem is that it is generally inexpensive to update a solution to an initial problem in order to solve another, slightly modified, problem. Thus incremental problems are an especially important class of problems when it is expensive to solve an initial problem, since this cost can be amortized over many, similar problems.

The class of incremental problems may lead to the development of practical solutions to the more difficult problems of moving and movable obstacles. Problems involving moving obstacles can be viewed as continuous versions of incremental problems, and problems with movable objects can be viewed as a combination of task-level and incremental problems.

Currently, there are no efficient algorithms for incrementally maintaining representations of configuration space subject to environmental changes. Currently, the only solution is to discard the old representation of the robot's configuration space and construct a new one, despite the fact that the new representation might differ only slightly from the original.

In this paper, we focus on the particular case of a line segment moving amidst polygonal obstacles, which are moved incrementally between planning problems. We call such problems "incrementally changing environments." A key contribution here is maintaining a representation of the robot's configuration space, rather than constructing the joint configuration space (that represents the simultaneous positions of all obstacles that might be moved). In general, we expect the changes to the environment to affect very little of the configuration space. Therefore our incremental algorithm should be very efficient on average.

In this paper, we assume a critical curve based exact cell decomposition as the underlying representation of configuration space. This choice reflects, in part, our desire to understand the deep issues of incremental maintainance, since this decomposition captures nicely the relationship between changes in the environment and changes in the configuration space. This choice of representation also reflects, in part, our desire for an efficient incremental algorithm. In other words, we are willing to pay a high initial cost, if the expected incremental cost to maintain the representation is low. There are many other representations that might be more efficient; but this is a good choice for a first study.

We note that the geometric robot motion planning litera-

*This work was supported by the National Science Foundation under grant number NSF-IRI-9110270.

†Department of Computer Science

‡Department of Electrical and Computer Engineering.

ture has primarily focused on planning for single problems, and not on reuse or modification of the costly representation. The geometric robot motion planning literature that does mention reusing the representation (*e.g.* [4]), mentions only the relative speed up of performing just the graph search without the additional cost of constructing the representation. In general, the time to construct the representation is only slightly worse than the size of the representation.

To the best of our knowledge, computing the all-pairs shortest paths trees has not been proposed in the geometric robot motion planning literature. This is probably because research has been focused on single planning problems, and because the cost of search is negligible relative to the cost of constructing a representation of $\mathcal{C}_{free}$. In the case of incrementally changing environments, the cost to incrementally maintain the representation of $\mathcal{C}_{free}$ is small. Therefore, the relative cost of search is increased, and it becomes important to reduce the search cost.

The remainder of this paper is organized as follows. In Section 2 we review the critical curve based exact cell decomposition of free space for a line segment translating and rotating amidst static polygonal obstacles (originally proposed by Schwartz and Sharir [29]). Section 3 discusses the need for a more explicit representation, and presents a plane sweep algorithm to compute this representation. Then, in Section 4, we briefly discuss how the explicit representation can be incrementally maintained. Section 5 gives our conclusions.

2 A Static Algorithm

This section reviews the representation used by the Schwartz and Sharir algorithm [29] for computing a collision-free trajectory for a line segment amidst polygons in the plane.[1] This is not the only exact representation of free space (*e.g.* [22, 34]); but we feel it is the best suited for a first examination of the problem of incrementally changing environments. The method of representation presented in this section can be generalized in a straight-forward manner to arbitrary polygonal robots moving amidst polygonal obstacles. In this case there are more types of contacts to consider [29]. The method can also be extended to other robots, as long as the robot and the obstacles have algebraic descriptions *i.e.* they are composed of algebraic surfaces of bounded complexity. This was done in [30] by means of Collins cylindrical algebraic decomposition.

The robot, which we shall denote by $\mathcal{A}$, is a line segment of length L, with endpoints P and Q. A configuration of $\mathcal{A}$ can be represented by the position $X = (x, y)$ of P, and the orientation θ of $\mathcal{A}$, which is the direction $\vec{PQ}$ with respect to the world coordinate frame. Any configuration for which $\mathcal{A}$ intersects some obstacle, $\mathcal{B}_i$, belongs to the configuration space obstacle $\mathcal{CB}_i$. The union of all $\mathcal{CB}_i$ is denoted by $\mathcal{CB}$. The complement of $\mathcal{CB}$ is the space of all collision-free configurations which we denote by $\mathcal{C}_{free}$. Given two configurations, q_{init} and $q_{goal} \in \mathcal{C}_{free}$, the geometric robot motion planner returns a function $\tau : [0,1] \to \mathcal{C}_{free}$ such that $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$.

[1]The notation follows [20].

2.1 Critical Curves and Noncritical Regions

In an exact cell decomposition for geometric robot motion planning, $\mathcal{C}_{free}$ is completely decomposed into a finite set of cells. In a critical curve based decomposition, the cells capture the interesting contacts between the robot and the environment. Since $\mathcal{C}_{free}$ is quite complex, the cells are generated by first projecting $\mathcal{C}_{free}$ onto $\mathbf{R}^2$, and then lifting the resultant regions into $\mathbf{R}^2 \times S^1$.

The boundary of $\mathcal{CB}$ is a finite set of ruled surfaces. Each surface is characterized by the set of environmental features in contact with the robot. The elementary contacts are of two types: (A) an obstacle vertex in contact with the interior of $\mathcal{A}$, and (B) a vertex of $\mathcal{A}$ in contact with an obstacle edge. When projected onto the xy-plane, these surfaces yield critical curves that intersect at critical points. The critical curves divide the plane into a finite set of noncritical regions. The noncritical regions are maximal open connected subsets of $\mathbf{R}^2$ such that the set of elementary contacts is invariant over the region. We exclude from the definition of noncritical regions any region contained within an obstacle. In other words, a critical curve is the locus of robot positions (the x, y coordinates of endpoint P) such that as P crosses the critical curve, the set of possible elementary contacts changes. There are six types of critical curves, which are described in detail in [29, 20]. Curves of types 1,2,4, and 5 are line segments, type 3 are circular arcs, and type 6 are positive portions of conchoids of Nichomedes.

2.2 Cell Formation

Each noncritical region specifies an open set of positions $X = (x, y)$ of P such that *some* orientation θ exists where the robot configuration $q = (X, \theta)$ of $\mathcal{A}$ is in $\mathcal{C}_{free}$. We call such orientations *free orientations.* These orientations determine legal ranges in the cylinder above the noncritical region, and thus define a noncritical cell in that cylinder.

The following is standard terminology for describing cells in the decomposition of $\mathcal{C}_{free}$. Let $F(X) = \{\theta : (X, \theta) \in \mathcal{C}_{free}\}$ be the set of all free orientations of $\mathcal{A}$ at the noncritical position X. The extremum of each interval, where $\mathcal{A}$ contacts some obstacle, is called a *limit orientation* of $\mathcal{A}$ at X. If the obstacles are in general position, then the limit orientation corresponds to a unique obstacle feature, called the *stop.* Define $\sigma(X)$ as the set of feature pairs (s_1, s_2) corresponding to the intervals in $F(X)$.

We can now define a cell in the cylinder above the noncritical region R as follows. Let R be a noncritical region and $(s_1, s_2) \in \sigma(X)$, where $X \in R$. Then cell(R, s_1, s_2) consists of all configurations $q = (X, \theta)$, such that $X \in R$ and θ is in the interval determined by X, s_1, and s_2. The set of cells as defined above partition $\mathcal{C}_{free}$ in the sense that the cells are disjoint, and the closure of their union is equal to the closure of $\mathcal{C}_{free}$.

2.3 The Connectivity Graph

Having described an exact cell decomposition of $\mathcal{C}_{free}$, we now describe the construction of the associated connectivity graph, which aids in the search for a collision-free trajectory of the robot. We denote the connectivity graph by $\mathcal{G}(V, E)$, where V is the set of vertices corresponding to the cells of the decomposition, and E is the set of edges such that two vertices are connected if and only if the corresponding cells are adjacent. Two cells are ***adjacent*** if and only if the base regions are adjacent and the set of orientations overlap everywhere on the common boundary. Given $\mathcal{G}$, planning consists of (1) determining the vertices v_{init} and v_{goal} associated with cells κ_{init} and κ_{goal} containing, respectively, the initial and goal configurations q_{init} and q_{goal}, (2) searching $\mathcal{G}$ for a path from v_{init} to v_{goal}, (3) extracting a trajectory from the corresponding sequence of cells.

The Schwartz and Sharir algorithm does not explicitly construct the noncritical regions and the cells above them [29]. Instead the algorithm associates with each curve a "left" region and a "right" region, and builds a connectivity graph based on these characterizations. Thus a region occurs as many times in the connectivity graph as there are curve segments on its boundary. Furthermore, the connectivity graph is not constructed in advance, rather it is generated as a by-product of the search process.

3 An Explicit Representaion

In Section 2, we described the noncritical regions in the plane that are the projections of the noncritical cells in the decomposition of $\mathcal{C}_{free}$ for a line segment robot. The Schwartz and Sharir algorithm does ***not*** create an explicit representation of the regions [29]. Instead, their algorithm performs $\mathcal{O}(n^4)$ intersection tests among the $\mathcal{O}(n^2)$ critical curves, where n is the number of obstacle edges and vertices. In their algorithm, each curve segment determines two regions, so there are $\mathcal{O}(n^4)$ noncritical regions, which produce $\mathcal{O}(n^5)$ cells in the connectivity graph.

This representation was not developed with the thought of incremental maintenance. There are no simple means to adjust the representation when the underlying environment is modified. We could keep track of which critical curve segments arise from which obstacle. Then, when an obstacle is moved, all associated curves and their intersections with other curves would need to be removed from the decomposition, and from the connectivity graph. New curves would be generated, intersections formed, and a new connectivity graph created.

Instead, if explicit regions are formed from the arrangement of critical curves, then local changes can be efficiently maintained through local operations to the representation. Although the regions are nontrivial [17], a plane sweep algorithm [10, 33] can be used to efficiently form the noncritical regions explicitly. This is possible due to the algebraic simplicity of the critical curves. There is a maximum degree of four for each curve, which implies a bounded number of intersections between any two curves. Therefore, an output-sensitive algorithm that computes all intersections in time $\mathcal{O}(k + n^2 \log n)$, where there are k intersections among $\mathcal{O}(n^2)$ critical curves, can be used.

As a by-product of region construction, a sample point for the region can be obtained. A sample point is useful for computing the set of stops for the region, as well as extracting a solution trajectory through the region. Since the sweep produces simple regions in sorted order, the connectivity of the regions is also a by-product. Furthermore, once a region has been formed, the cells contained in the cylinder above the region can be formed by computing the legal ranges of orientation (the stops) when the vertex P of the line segment is in the region (at the sample point). To test whether two cells are adjacent, it must be determined that the two cells share a critical curve section and that their range of orientations overlap on the boundary. Determining whether two cells share a critical curve section is equivalent to testing region adjacency; and determining orientation overlap is simply comparing stop values. Thus the connectivity graph can also be obtained as a by-product of the plane sweep algorithm.

When obstacles are incrementally moved and noncritical regions change, the connectivity graph of the regions may also change. The local changes to the region connectivity graph can be used to propagate the changes to the cells above each region. Because the regions and their relationships are explicit, direct manipulation, without the need to search, is sufficient to maintain the geometry and topology of the representation.

The rest of this section is as follows. In Section 3.1 we present a hierarchical representation for the cells. A hierarchical representation is central to efficient maintenance of the underlying representation of $\mathcal{C}_{free}$. Section 3.2 provides the rationale for computing the all-pairs shortest paths trees.

3.1 A Hierarchical Representation

This section examines the problems that incrementally changing environments present for the maintenance of the critical curves. In particular, we comment on some ramifications of using hierarchical bounding approximations of noncritical cells as an underlying data structure.

Considerable efficiency can be gained from hierarchically representing obstacles and their associated critical curves, noncritical regions, and noncritical cells. Hierarchical bounding boxes have been used effectively in solid modeling and graphics *e.g.* [28, 24]. The use of bounding boxes limits the number of intersection tests between object primitives and the number of intersections needed for rendering.

Critical curves of Types 1 through 4 characterize the limiting positions of the robot as it navigates in close proximity of a convex obstacle. We note that these curves are all contained within a bounding polygon that has the same shape as the obstacle polygon, but is "grown" radially by distance L. Similarly, curves of Types 3 and 4 are contained within a circle of radius L centered at each convex obstacle vertex. Together, the Type 2 and (portions of the) Type 3 curves yield a generalized polygon that completely encloses all critical curves local to the obstacle. Furthermore, curves

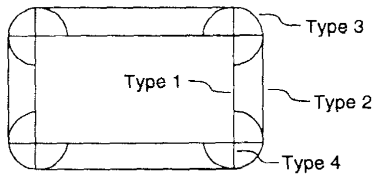

Figure 1: Critical curves associated with a rectangle.

of Type 5 and 6 are also contained within a circle of radius L centered at a convex obstacle (generating) vertex. Curves of Type 6 not only are contained within the circle centered at the generating vertex, but also are bounded from above by the Type 2 curve of their generating edge, and from below by a line parallel to the generating edge and passing through the generating vertex. Curves of Types 1-4 for a rectangle are illustrated in Figure 1.

When the environment consists only of convex polygons, then only critical curves of Types 5 and 6 may be candidates for creation and destruction when obstacles are moved, as curves of Types 1 through 4 are permanently associated with each individual obstacle. Type 5 curves only occur when two obstacle vertices come within distance $d < L$ from each other; and Type 6 curves only occur when a vertex of one obstacle comes within distance L of another obstacle (where L is length of the line segment). The interesting extra curves are the conchoids of Nichomedes, which are enclosed in the bounding box $[[x = -x_m, x = x_m], [y = 0, y = L - d]]$, where x_m corresponds to $y_m = (L^2d)^{1/3} - d$. This box is specified in the local coordinate frame of the conchoid. The base of the box is parallel to the generating edge. Thus, unless the environment is exceptionally cluttered, so that there are few obstacles within distance L of one another, bounding approximations are sufficient to detect that most obstacle motion produces no effect on the topology of the set of critical curves.

3.2 Representation Reuse

Here, we examine the costs and benefits of computing the all-pairs shortest paths trees in the connectivity graph of the exact cell decomposition of $\mathcal{C}_{free}$ for planning. By computing the all-pairs shortest paths trees in the connectivity graph of $\mathcal{C}_{free}$, planning is reduced to lookup. The time required to compute the all-pairs shortest paths trees (with a simple algorithm) is $\mathcal{O}(|V||E|\log|V|)$ where E and V are the sets of edges and vertices in the connectivity graph. (We conjecture that the connectivity graph is sparse in practice so $|E| = \mathcal{O}(|V|)$.) The space required for the all-pairs shortest paths trees is $\mathcal{O}(|V|^2)$.

If many problems are to be solved in the same static environment, then the time to compute the all-pairs shortest paths trees can be amortized over all planning problems. If the savings in the time to compute a solution is substantial, the space cost may be affordable. For example, if $\mathcal{O}(|V|)$ problems are solved, then approximately the same amount of time will be expended on individual searches as will be spent on computing the all-pairs shortest paths trees. If more than $\mathcal{O}(|V|)$ problems are solved, then computing the all-pairs shortest paths trees may have consumed less time than all individual searches combined. All of the individual problems require only $\mathcal{O}(|V|)$ additional space for the search (A* search essentially computes a single-source shortest paths tree) instead of $\mathcal{O}(|V|^2)$ space for the all-pairs shortest paths trees. The crucial difference between separate search efforts and computing the all-pairs shortest paths trees is that the all-pairs shortest paths trees are computed off-line, providing the user with real-time solutions. For the problem of incrementally changing environments, the all-pairs shortest paths trees must be incrementally maintained. On average we expect that the all-pairs shortest paths trees can be maintained in much less time than it takes to search a connectivity graph.

Note that it may be more efficient to build up to the all-pairs shortest paths trees as required for problem solving (*i.e.* maintain only the set of single-source shortest paths trees computed so far) since the distribution of planning problems may not uniformly cover the configuration space. A review of research on maintaining a set of single-source shortest paths trees is given in [26].

4 Incremental Maintenance

In this section we present an algorithm for incrementally maintaining the exact cell decomposition of $\mathcal{C}_{free}$ presented in Section 3, given incremental changes to the environment. Small changes in the environment require corresponding small changes in the representation. Section 4.1 gives an overview of incremental graph algorithms. Efficiently maintaining the all-pairs shortest paths trees reduces planning to lookup. Then in Section 4.2 we describe the combined effects of explicit hierarchical bounding approximations and the all-pairs shortest paths trees.

4.1 Incremental Graph Algorithms

This section gives an overview of incremental graph algorithms, and how they are incorporated into our solution for planning in incrementally changing environments.

An incremental graph algorithm is given as input a graph, a subgraph that satisfies some desired property, and a modification to the graph. The algorithm outputs a new subgraph that satisfies the desired property in the modified graph. We say that a vertex is "affected" by the change if it changes its local property from the original graph to the modified graph. The minimum amount of work any incremental algorithm must perform is to correct every affected vertex. The complexity of an incremental graph algorithm should, therefore, be a function of the size of the set of affected vertices. A graph algorithm is said to be bounded incremental if it runs in time that is polynomial in the size of this set [26]. Algorithms that always check every vertex are therefore unbounded incremental. An incremental algorithm is designed to take advantage of small modifications to an underlying graph. On difficult problems (where most, if not all, vertices are affected), it may be faster to recompute the subgraph.

However, on average problems, incremental algorithms can be expected to run much faster [26, 5].

Incremental algorithms play important roles in the simplex method [1] and in areas such as network flows, matching, and computational circuit analysis [21, 16, 2]. We have shown that hierarchical search can be efficiently implemented by using an incremental variant of Dijkstra's algorithm for maintaining a single-source shortest paths tree in a dynamic graph [5].

There is a considerable amount of research dedicated to the dynamic all-pairs shortest paths problem [27, 14, 3, 23, 26, 19]. The all-pairs shortest paths problem is of interest if we want to consider many different planning problems within the same environment. We expect that incrementally maintaining the all-pairs shortest paths trees is more efficient than computing a single-source shortest paths tree for each new problem. Unfortunately, most of this research examines single changes to the graph, for example the insertion of an edge. For the geometric robot motion planning application, vertices are generally added and deleted, which changes many edges simultaneously. The only such algorithm known to us is by Ramalingam and Reps [26].

4.2 Combining Hierarchy and Shortest Paths

In this section we discuss potential synergies between the explicit hierarchical representation of noncritical cells and the incremental maintenance of the all-pairs shortest paths trees.

4.2.1 Deferring Updates

By hierarchically maintaining the noncritical regions, *e.g.* by a succession of bounding approximations, we can effectively limit the number of complex intersection tests that must be performed as an obstacle moves close to another. For example, a plane sweep within a very small area can be performed. By analyzing the nature of the intersection, the types of changes that have occurred can be assessed. Example changes include which vertices of the connectivity graph of C_{free} have changed (their underlying region has new geometry or requires subdivision), and whether or not this change is significant.

Small obstacle motions may leave the topology of the noncritical regions unchanged. In other words, the connectivity graph and embedded all-pairs shortest paths trees remain correct. Therefore the shortest path between two vertices in the graph is still available. Thus the solution to a specific problem can be found without having the precise geometry of all regions available. In the event that no affected vertices are used in the solution sequence, a solution trajectory can be generated from the solution sequence and the actual repair to the geometry of the affected regions can be deferred. This ability to defer certain operations adds flexibility to the planner, and allows for parallelism. This is to say that the process for computing a solution trajectory can safely run at the same time as the process for repairing cells.

4.2.2 Assessing Cell Cost

A hierarchical representation can be used to induce meta-cells, which can be assessed a cost that reflects the complexity of computing a trajectory through the underlying cells. For example, a "cluster" of noncritical regions can be grouped together, say those in the intersection of two bounding approximations. All of the cells above these regions can similarly be aggregated together and treated as a whole. This has much the same flavor of the hierarchical approximate cell decomposition algorithm in which a MIXED cell is subdivided into many subcells, changing the connectivity graph in a local area [5]. Here, regions and cells are grouped together, as well as subdivided. Grouping cells has the additional benefit of effectively decreasing the size and simplifying the connectivity graph.

So, when an obstacle is moved, the approximating representations are analyzed to assess the cost to repair the connectivity graph. If there are many new regions to be formed, or many intersections are required, then the cells can be aggregated together, and the corresponding vertices in the connectivity graph replaced by a single vertex with high cost. This local change to the connectivity graph is then propagated to the all-pairs shortest paths trees. If the optimal path avoids the high cost vertex, then examining the underlying changes can be deferred. Otherwise, the underlying region must be subdivided, and the high cost vertex replaced in the connectivity graph by the new vertices corresponding to the new regions. Again, this local change to the connectivity graph is propagated to the all-pairs shortest paths trees and the solution extracted.

The amount of work involved to repair the data structures given some obstacle has been moved can be assessed at a fairly high level of abstraction. The intersection of bounding approximations may indicate, for example, that a great many changes are likely without actually examining any specific change. Such assessments can be used to tradeoff the quality of solution for planning time. For example, the estimated cost to repair the connectivity graph can be charged to the cells involved. This change in cell costs will be propagated to the all-pairs shortest paths trees.

4.2.3 Fast Suboptimal Solution Paths

The all-pairs shortest paths trees provide the path cost of the least cost path between every pair of vertices in the graph. In general, there are multiple least cost paths. Similarly, there might be many paths for which the path costs differ by ϵ. If the cost of an edge changes by ϵ, it is possible that a large number of vertices will be affected. If the all-pairs shortest paths trees are maintained only to within ϵ accuracy, then small changes in vertex costs may not affect any other vertices. This approach might save a great deal of tree maintenance, but it comes at the expense of suboptimal solution paths. However, we can quantify the extent to which a path is suboptimal. In particular, we can guarantee that the cost of the final solution path will be no more than $(1+\epsilon)$ times the cost of the least cost path. A similar result is obtained in [19] and mentioned in [8].

5 Conclusions

In this paper we discuss the concept of incremental algorithms to solve problems in geometric robot motion planning where the input changes between planning problems. We adopt the critical curve based exact cell decomposition of [29] as our basic representation. We extend the underlying representation to accommodate incrementally changing environments. The novel contribution to geometric robot motion planning is to efficiently maintain the representation of the robot's free space subject to incrementally moving obstacles.

A key improvement to [29] is a plane sweep to explicitly form the noncritical regions. A by-product of this is a connectivity graph of the regions. A hierarchy of bounding approximations is associated with the obstacles and the local noncritical cells. This enables local reasoning about local changes. It also enables efficient aggregation of cells to defer reasoning or to reason at an abstract level.

The changes in the environment are expected to be local to a small area. The affected regions can be repaired, and the changes can be propagated to the cells above the regions. The change in cells affects the connectivity graph. Then the all-pairs shortest paths trees embedded in the connectivity graph must be updated. We expect that on average the overhead of this incremental maintenance will be quite small. The presence of the all-pairs shortest paths trees, and an incremental algorithm to maintain them, results in a planner that is capable of reacting quickly to obstacles that have been moved since plan generation. The ability to efficiently maintain a representation of $\mathcal{C}_{free}$ subject to obstacles that are incrementally moved may lead to efficient algorithms for the harder problems of moving obstacles and movable objects.

References

[1] R. Ahuja and J. Orlin. The scaling network simplex algorithm. *OR*, 40(1):S5–S12, 1992.

[2] B. Alpern, R. Hoover, B. Rosen, P. Sweeney, and F. Zadeck. Incremental evaluation of computational circuits. In *Proc ACM-SIAM Symp on Discrete Algs*, 1990.

[3] G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *J Algorithms*, 12, 1991.

[4] F. Avnaim, J. D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1988.

[5] M. Barbehenn and S. A. Hutchinson. Efficient search and hierarchical motion planning by dynamically maintaining single-source shortest paths trees. *IEEE Trans on Robotics and Automation*, 11(2), 1995.

[6] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[7] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int'l J. of Robotics Research*, 10(6):628–649, 1991.

[8] D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

[9] S. Buckley. Fast motion planning for multiple moving robots. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1989.

[10] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments. In *Proc IEEE Symp on Found of Comput Sci*, 1988.

[11] P. Chen and Y. Hwang. Practical path planning among movable obstacles. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1991.

[12] B. Dacre-Wright, J.-P. Laumond, and R. Alami. Motion planning for a robot and a moving object amidst polygonal obstacles. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1992.

[13] M. Erdmann and T. Lozano-Perez. On multiple moving objects. Technical Report 883, MIT AI Lab, 1986.

[14] S. Even and H. Gazit. Updating distances in dynamic graphs. *Methods of Operations Research*, 49, 1985.

[15] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using the relative velocity paradigm. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1993.

[16] D. Goldfarb, J. Hao, and S. Kai. Efficient shortest path simplex algorithms. *OR*, 38(4), 1990.

[17] D. Halperin and M. Sharir. Arrangements and their applications in robotics. In *Workshop on the Algorithmic Foundations of Robotics*, San Francisco, February 1994.

[18] K. Kant and S. Zucker. Toward efficient trajectory planning: Path velocity decomposition. *Int'l J. of Robotics Research*, 5, 1986.

[19] P. Klein and S. Sairam. A fully dynamic approximation scheme for all-pairs shortest paths in planar graphs. In *LNCS 709*. Springer-Verlag, Berlin, 1993.

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

[21] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[22] D. Leven and M. Sharir. An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. In *Proc ACM Symp on Comp Geom*, 1985.

[23] C. C. Lin and R. C. Chang. On the dynamic shortest path problem. *Journal of Information Processing*, 13(4), 1990.

[24] M. Mantyla and M. Tamminen. Localized set operations for solid modeling. In *Proc. SIGGRAPH*, July 1983.

[25] P. O'Donnell and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1989.

[26] G. Ramalingam and T. Reps. Bounded incremental computation. Technical Report 1172, Dept of Comp Sci, Univ of Wisc at Madison, 1993.

[27] H. Rohnert. A dynamization of the all pairs least cost path problem. In *LNCS 182*. Springer-Verlag, Berlin, 1985.

[28] Scott D. Roth. Ray casting for modeling solids. *Comp Graphics and Image Proc*, 18, 1982.

[29] J. Schwartz and M. Sharir. On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. In J. Schwartz, M. Sharir, and J. Hopcroft, 1987.

[30] J. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. In J. Schwartz, M. Sharir, and J. Hopcroft, 1987.

[31] J. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. In J. Schwartz, M. Sharir, and J. Hopcroft, 1987.

[32] J. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry, and Complexity of Robot Motion.* Ablex, Norwood, NJ, 1987.

[33] I. Shimshoni and J. Ponce. Finite resolution aspect graphs of polyhedral objects. In *Proc IEEE Workshop on Qualitative Vision*, 1993.

[34] S. Sifrony and M. Sharir. A new efficient motion-planning algorithm for a rod in two-dimensional polygonal space. *Algorithmica*, 2, 1987.

[35] P. Tournassoud. A strategy for obstacle avoidance and its application to multi-robot systems. In *Proc IEEE Int'l Conf on Robotics and Automation*, 1986.