SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NEW YORK 14853

TECHNICAL REPORT NO. 838

March 1989

ON MONOTONE PATHS AMONG OBSTACLES, WITH APPLICATIONS TO PLANNING ASSEMBLIES

By

Esther M. Arkin
Robert Connelly†
Joseph S. B. Mitchell

This paper to appear in the *Proceedings of the Fifth Annual ACM Symposium on Computational Geometry*, Saarbrücken, West Germany, June 5-7, 1989.

† Department of Mathematics, Cornell University

ON MONOTONE PATHS AMONG OBSTACLES, WITH APPLICATIONS TO PLANNING ASSEMBLIES

(extended abstract)

Esther M. Arkin†
Robert Connelly‡
Joseph S. B. Mitchell*

Cornell University
Ithaca, NY 14853

Abstract

We study the class of problems associated with the detection and computation of monotone paths among a set of disjoint obstacles. We give an $O(nE)$ algorithm for finding a monotone path (if one exists) between two points in the plane in the presence of polygonal obstacles. (Here, E is the size of the visibility graph defined by the n vertices of the obstacles.) If all of the obstacles are convex, we prove that there always exists a monotone path between *any* two points s and t. We give an $O(n \log n)$ algorithm for finding such a path for any s and t, after an initial $O(E + n \log n)$ preprocesing. We introduce the notions of "monotone path map", and "shortest monotone path map" and give algorithms to compute them. We apply our results to a class of separation and assembly problems, yielding polynomial-time algorithms for planning an assembly sequence (based on separations by single translations) of arbitrary polygonal parts in two dimensions.

†Partially supported by NSF Grants DMC-8451984 and ECSE-8857642.

‡Partially supported by NSF Grant MCS-790251.

* Partially supported by a grant from Hughes Research Laboratories, Malibu, CA, and by NSF Grants IRI-8710858 and ECSE-8857642.

1. Introduction

In this paper, we study the problem of finding a *monotone* path from a point s to a point t among a set of disjoint obstacles in the plane. We say that a path is monotone with respect to a direction vector d (or, *d-monotone*) if, for every pair of points u and v on the path such that u precedes v, the vector uv has a nonnegative inner product with d.

The notion of monotonicity has been prevalent in computational geometry. *Monotone polygons* have considerable special structure which allows, for instance, a simple linear-time triangulation procedure. It is known ([PS]) that a simple polygon can be tested for monotonicity in linear time. In fact, within the same time bound, one can determine the entire set of directions (which will be a convex cone) along which the polygon is monotone.

We assume that the domain space Ω is given to us as a (possibly unbounded) simple polygon with k disjoint simple polygonal holes ("obstacles"). Let n be the total number of vertices defining Ω.

There are two questions of immediate interest: (i). If a direction d is given to us, can we determine whether or not there exists a monotone path from s to t, and, if so, produce one such path? and (ii). If no direction is specified ahead of time, can we determine whether or not there exists a direction d and a path from s to t such that the path is monotone with respect to d?

Figure 1.1 shows a path which is monotone with

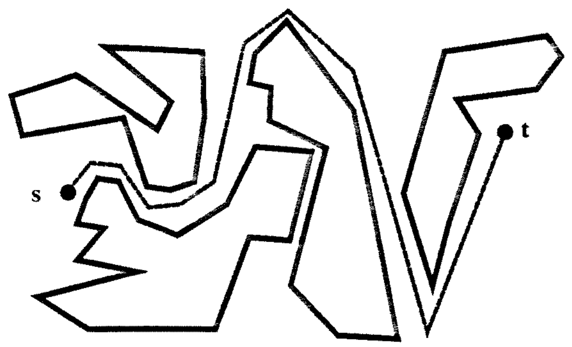

Figure 1.1. A monotone path in the x-direction.

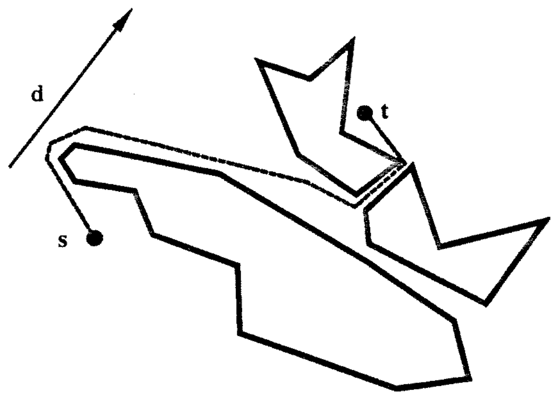

Figure 1.2. d-monotone path, no x-monotone path.

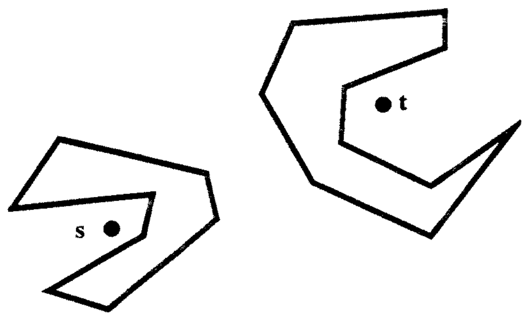

Figure 1.3. No monotone path in any direction.

respect to the x-axis. Figure 1.2 shows an example of a case in which there is no path from s to t monotone with respect to the x-axis, but there is a monotone path with respect to the direction d. And, Figure 1.3 shows an example of a case in which there is *no* direction d with respect to which there exists a monotone path.

A monotone path is one which does not "double back", so it describes a path which, in a certain sense, requires little turning. We were motivated to study this class of problems by several questions, including: 1). How can we find paths which are "nice" in some sense? 2). If we are charged only for motion in some given direction (say, in direction d), then how do we find a path from s to t which minimizes this motion? Clearly, if there exists a path which is monotone with respect to d, then this path will minimize motion in direction d.

An important application of our results is to the separability problem: Given a set of parts in the plane, how do we disassemble them (or, equivalently, how can we assemble them)? Using the results on the computation of monotone paths, we provide a polynomial-time algorithm for determining a sequence of motions to assemble a set of arbitrary polygonal parts in the plane, under the following assumptions: There is one "hand" that can translate any subset of parts in any one direction. (No rotations are allowed.) We disassemble the set S of parts by partitioning S into two disjoint nonempty subsets (S_1 and S_2) of parts by translating one subset (S_1) in some direction d a sufficient distance so that the problems of disassembling S_1 and S_2 are now independent. We then repeat the procedure on the subproblems defined by S_1 and S_2, continuing to disassemble sets of parts until each set is a singleton.

Our results are summarized as follows:

1). We answer the question (i) above with a simple $O(n \log n)$ algorithm to find a monotone path (if it exists) in a given direction d. We also prove a lower bound of $\Omega(n \log n)$ (even if the obstacles are all convex), assuming that we are required to output a monotone path. In fact, we give an $O(n \log n)$ algorithm which solves a slightly more general problem: find a path P which minimizes the length of the path P_d that we get if we project P orthogonally onto a line in direction d.

2). We answer question (ii) above with an $O(nE)$ algorithm to find a monotone path (if it exists) in *any* direction (here, E is the size of the visibility graph defined on the set of obstacles). (In the special case of convex obstacles, we get an improved bound of $O(E + n \log n)$.) We characterize the set of all directions along which there exists a monotone path from s to t, prove a bound of $\Theta(k)$ on its combinatorial size, and show how to compute it within our stated time bounds. Within the same time bounds, we construct a *monotone path map* (MPM), which is a subdivision of Ω according to the structure of the set of monotone paths from a given source point s to any other point t in Ω.

3). In the case of convex obstacles, we prove a remarkable theorem that there will *always* exist a monotone path from s to t in *some* direction. We show that the set of all directions along which there is a monotone path from s to t is a convex cone. We provide an $O(E + n \log n)$ algorithm that produces a monotone path map (with respect to s), allowing us to produce the cone of directions for which a monotone path exists from s to any query point t in $O(\log n)$ time. Furthermore, after locating t within the MPM, we can output a monotone path to t in time proportional to its size.

4). We also examine the special case of monotone paths in a simple polygon (without holes), and we show that the (possibly empty) set of all possible directions along which monotone paths exist can be found in time $O(T + n)$ (where T is the time to triangulate the simple polygon) for a given s and t, or, with the same amount of preprocessing, we can answer two-point queries for any given s and t in time $O(\log n)$, returning the valid interval of directions along which there is a monotone path from s to t.

5). Finally, we show how to apply some of our results to yield efficient algorithms to plan assemblies of planar parts. Our algorithm determines a direction d such that some (nontrivial) subset of the parts can be translated along d away from the remaining parts (or determines that no such separation is possible). By repeated applications of this technique, we arrive at a means of planning disassemblies (and hence assemblies) of arbitrary polygonal parts.

2. Min Travel in a Given Direction

The problem of finding shortest paths in three dimensions has been shown to be NP-Hard [CR]. We may ask if using the distance function $\sqrt{x^2 + y^2}$ rather than the usual Euclidean metric, $\sqrt{x^2 + y^2 + z^2}$, makes the problem any easier. However, the construction of [CR] suggests that this is not the case – it remains NP-Hard even if we are not "charged" for travel in the z-direction.

In the two-dimensional case, the current best algorithms for computing Euclidean shortest paths among polygonal obstacles require quadratic time. If, however, we measure distance according to the metric $\sqrt{x^2} = |x|$ rather than the Euclidean metric ($\sqrt{x^2 + y^2}$), we obtain the problem of minimizing the length of a projected path along a given direction d, where, without loss of generality, d lies along the x-axis.

Our algorithm is based on searching a graph which is constructed by a straightforward application of the plane sweep paradigm. (We give only a sketch here.) We sweep the plane with a vertical line, building the *vertical adjacency map* (which connects each vertex to the first obstacle boundary point hit when going up from the point, and when going down, resulting in a trapezoidization of Ω). The resulting diagram consists of a planar subdivision with $O(n)$ vertices and line segments, and it defines a planar graph (the *vertical adjacency graph* (VAG)) whose edges are defined by the line segments. See Figure 2.1. We assign the length of an edge to be the length of its projection onto the x-axis. The problem of finding an x-minimizing path from s to t is then solved by running Dijkstra's algorithm on the resulting planar graph.

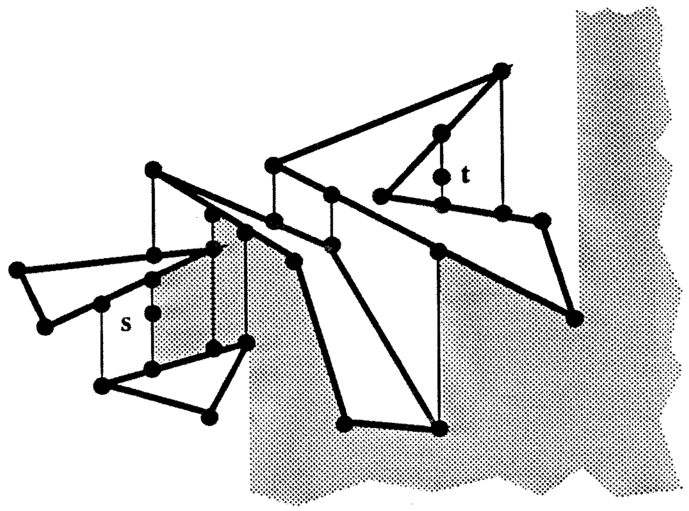

Figure 2.1. A vertical adjacency graph.

Theorem 2.1 *We can find a path which minimizes the length of its projection along any given direction in time $O(n \log n)$.*

Note that if the length of the projected path is equal to the projected distance between s and t, then the path is in fact monotone. Also note that, given the VAG, it is easy to construct the set of all points t which can be reached by a monotone path (monotone, that is, in the given direction) from s. (Refer to the shaded region in Figure 2.1.) Thus, we have:

Corollary 2.2 *One can detect whether or not there exists a monotone path from s to t in a fixed direction*

d in time $O(n \log n)$, and, if so, output one such path within this same time bound. Within the same time bound, one can find the region of all points t for which there exists a d-monotone path from s to t.

We also have the following lower bound result.

Theorem 2.3 *A lower bound on the complexity of finding a path (if one exists) from s to t which is monotone in a given direction d is $\Omega(n \log n)$. (This assumes that we must actually output an ordered list of turning points in a piecewise-linear representation of the path.)*

Proof: The simple reduction is from sorting, and is omitted here. ▮

3. A General Algorithm

We turn now to the problem of finding a monotone path in *any* direction. More precisely, we want an algorithm to find a direction d (if one exists) such that there exists a d-monotone path from s to t. Further requirements of our algorithm may be to produce a description of the set of all d's for which a monotone path exists and/or to produce a representation of a d-monotone path.

We begin with some basic properties of monotone paths.

Lemma 3.1 *If a d-monotone path is pulled taut, then it will remain d-monotone.*

Corollary 3.2 *If a d-monotone path from s to t exists, then there is a d-monotone path from s to t which is taut, and is therefore piecewise-linear and lies on the visibility graph.*

Corollary 3.3 *If there is a d-monotone path from s to t for some d, then there is a d_e-monotone path from s to t, where d_e is a vector orthogonal to an edge e of the visibility graph.*

Remark: A further corollary of Lemma 3.1 is the result (4) of Section 1, which has to do with the special case of monotone paths within simple polygons. By combining Lemma 3.1 with the results of [GH] (on two-point shortest path queries in simple polygons) and [PS] (on finding cones of monotone directions for simple polygons), we obtain the claimed results.

Thus, it suffices for our algorithm to check only those E directions d_e defined by edges of the visibility graph.

Lemma 3.4 *The set of all directions d for which a monotone path exists from s to t consists of a union of at most k convex cones whose bounding rays are orthogonal to edges of the visibility graph.*

Proof: (Sketch) We show that between any two taut paths that are monotone in different directions there must be an obstacle. ▮

In fact one can exhibit an example to show that the above bound is tight, as shown in Figure 3.1, where k (nonconvex) obstacles give rise to k "skinny" cones of feasible directions d for which there exists a d-monotone path.

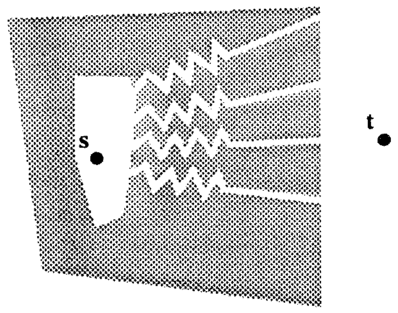

Figure 3.1. k cones of monotone directions.

The above lemmas justify the following algorithm.

1). Build the visibility graph (VG) of s, t, and the obstacle space Ω. This can be done in $O(E + n \log n)$ time and $O(E)$ space [GM].

2. Sort the edges of VG according to slope. This requires time $O(E \log E) = O(E \log n)$.

3. For each direction d_e that is orthogonal to some edge e of VG (there are two such directions for each e), orient the edges of the visibility graph according to d_e, thereby obtaining a directed graph DVG(d_e). The naive means of doing this requires $O(E)$ time per direction d_e. However, the fact that we have the edges e sorted by slope allows us to do this step faster by noting that if we proceed through the directions d_e in slope order, then the DVG changes in a very small way from one direction to the next; namely, one edge of DVG gets flipped at each step. This allows the orientation to be done in constant amortized time per direction, for a total cost of $O(E)$.

4. Now, to find a monotone path from s to t (if one exists) requires that we search the directed graphs DVG(d_e) for a directed path from s to t. This can be

done, for instance, by depth-first search in linear time (linear, that is, in the number of edges in the graph). Hence we get a total time bound of $O(E^2) = O(n^4)$.

We get an improvement to the above naive algorithm by reducing the size of the graph that we search. Instead of rotating through the directions d_e searching DVG(d_e) (which is of size E), we can rotate through the directions d_e, keeping track of the way in which the *directed vertical adjacency graph* DVAG(d_e) (of size $O(n)$) changes. (The directed vertical adjacency graph is simply the oriented version of the VAG, in which each edge is oriented according to d_e.) Again, one can show that the graph DVAG(d_e) changes in a trivial way from one direction to the next. The result is the following:

Theorem 3.3 *Given a polygonal obstacle environment of size n, we can detect the existence of a monotone path from s to t in time O(nE), and we can output such a path if it exists within this same time bound. In fact, within this same time bound, we can construct the set of all directions d along which there is a monotone path from s to t.*

The bottleneck in both the naive solution and its improvement is the search step, which must be done for each direction d_e. It is quite plausible that a better method could be employed that would use the fact that as we proceed from one directed graph to the next, the number of flips is small (constant). It may be possible to reuse search information from one step to the next.

4. Convex Obstacles

When all of the obstacles are convex, and the bounding simple polygon is also convex, then we will show that there *always* exists a monotone path from s to t. We originally found this fact to be quite remarkable, as it seemed that by building "spirals" out of disjoint line segments, and cluttering the space with enormous numbers of obstacles, it should be possible to prevent a monotone path. However, as our theorem shows, such is not the case.

We begin with a few definitions. For any direction d, let $d_\perp$ be the vector d rotated counterclockwise by $\pi/2$. We define the *leftmost path in direction d* out of s, $l(d)$, as the semi-infinite path starting from s constructed in the following manner: Shoot a ray from s in direction $d_\perp$. If it goes to infinity (or hits the outer boundary of Ω) without hitting an obstacle, then this ray is $l(d)$. Otherwise, it will hit an obstacle, O, at some point p. The path $l(d)$ proceeds by turning right at point p, following the boundary of O until it can again proceed in direction $d_\perp$. This construction continues until the path either goes off to infinity or hits the outer boundary of Ω. Refer to Figure 4.1. We similarly define the *rightmost path in direction d* out of s, $r(d)$, by shooting a ray in direction $-d_\perp$ and turning left every time we hit an obstacle.

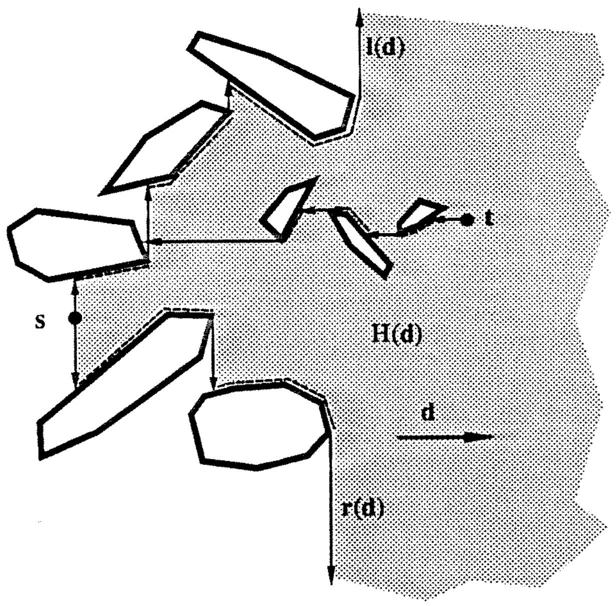

Figure 4.1. Defining $l(d)$, $r(d)$, $H(d)$, waterfall paths.

If two paths P_1 and P_2 are monotone in direction d and do not *cross* each other (although they may coincide along subpaths), then we say that P_1 is *left* of P_2 if at the first point where the paths diverge P_1 lies left of P_2. (Note that when they first diverge they cannot split apart by more than π, since they are both monotone in direction d; thus, the notion of being "left" at the point of divergence is well-defined.) Another way to say this is to define the polygon between P_1 and P_2 as that which lies to the right of the path from s along P_1 to the outer boundary (or the circle at infinity if none), and then returns to s along the reverse of P_2. We similarly define the notion of one path being *right* of another.

Lemma 4.1 *The paths l(d) and r(d) intersect only at point s. In fact, l(d) and r(d) cannot hit the same obstacle.*

Lemma 4.2 *The path l(d) (resp., r(d)) is monotone* *in direction d, and is the leftmost (resp., rightmost) such path.*

By Lemma 4.1, the paths $l(d)$ and $r(d)$ together form a simple path $\mathcal{P}(d)$ defined by traversing $r(d)$ in reverse until we reach point s and then by traversing $l(d)$ from s. The path $\mathcal{P}(d)$ partitions Ω into two regions, $H(d)$ and $H^c(d)$, where $H(d)$ is the set of points which lie to the right of $\mathcal{P}(d)$. Refer again to Figure 4.1, where $H(d)$ is shown shaded.

Lemma 4.3 *There exists a d-monotone path from s to any point $t \in H(d) \cap \Omega$.*

Proof: Construct a path P from t to s as follows: go in direction $-d$ from t until either an obstacle is hit or until $\mathcal{P}(d)$ is hit. If we hit $\mathcal{P}(d)$, we are done, since both $l(d)$ and $r(d)$ are monotone in direction d. Otherwise, we must hit an obstacle, in which case we continue around the boundary of the obstacle in the direction that keeps us monotone in direction $-d$ (this is always possible by the convexity of the obstacles). We leave the boundary of an obstacle at the first chance we have to proceed in direction $-d$. This process continues until we hit $\mathcal{P}(d)$. Refer to Figure 4.1. Clearly, the path P that we construct in this way is monotone in direction $-d$ (and thus its reverse is monotone in direction d). We call the resulting path P a "waterfall path", since it follows the path that water would follow from t if gravity were in direction $-d$. ▮

Lemma 4.4 *$H(d)$ is precisely the set of all points which are reachable by a d-monotone path from s.*

Proof: By the previous lemma, it suffices to show that it is not possible to reach a point $t \in H^c(d)$ along a d-monotone path. Consider any such t and assume that P is a d-monotone path from s to t. P must enter the open halfplane defined by s and d. (P can never enter the open halfplane defined by s and $-d$.) Thus, P will have to enter $H(d)$. But since t is not in $H(d)$, P will have to cross $\mathcal{P}(d)$, contradicting Lemma 4.2. ▮

Theorem 4.5 *If all obstacles are convex, then the set of all directions d for which a monotone path exists from s to t is nonempty.*

Proof: (Sketch) Our goal is to prove that there exists a direction d such that $t \in H(d)$. The basic idea is to imagine what happens to $H(d)$ as we rotate d counterclockwise. For simplicity, let us assume that no three vertices are colinear. We keep track of the paths $l(d)$ and $r(d)$. The only directions d which crucially affect $l(d)$ or $r(d)$ are those directions which are orthogonal to a visibility graph edge (in the reduced visibility graph that consists only of those visible edges which lie on lines of tangency between two obstacles). At any such critical direction d, one of $l(d)$ or $r(d)$ makes a discontinuous jump, as shown in Figure 4.2.

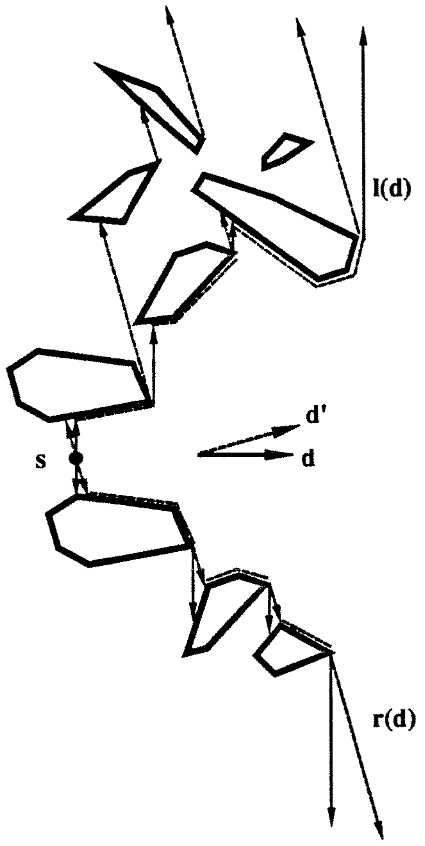

Figure 4.2. Sweeping the paths $l(d)$ and $r(d)$.

Assume that $l(d)$ makes a discontinuous jump from $l(d^-)$ to $l(d^+)$. (The non-degeneracy assumption implies that $r(\cdot)$ varies continuously at d.) Then, every point t in the region between the two paths $l(d^-)$ and $l(d^+)$ is reachable from s along a d^+-monotone path, since the region between the two paths belongs to $H(d^+)$. Thus, there are no points "missed" when d passes through a critical value, and everything varies continuously otherwise; hence, every point t must lie within *some* $H(d)$. ▮

Corollay 4.6 *Given a planar graph with an embedding such that every face is convex, there exists a monotone path between any two nodes of the graph. (The outer face must be convex in the sense that the outermost cycle forms a convex polygon.)*

Corollary 4.7 *Given a collection of arbitrary convex bodies in $\Re^m$ ($m \geq 2$) with disjoint interiors, and given any two points s and t not in the interior of the obstacles, there always exists a monotone path (in some direction) from s to t which avoids the obstacle interiors.*

Proof: Consider any plane through s and t, and restrict the path to lie on this plane. This reduces to the two-dimensional case proven in Theorem 4.5. ∎

Theorem 4.8 *If all obstacles are convex, then the set of all directions d for which a monotone path exists from s to t consists of a single (nonempty) convex cone.*

Proof: (Sketch) Assume that there exist monotone paths from s to t in directions d_1 and d_2 ($\neq d_1$). Thus, $t \in H(d_1)$ and $t \in H(d_2)$. Assume that d_1 lies to the left of d_2, meaning that $d_2 \times d_1 > 0$, and let d be any direction in the convex cone defined by d_1 and d_2. We must show that $t \in H(d)$. One can show this by arguing that $l(d)$ lies between $l(d_1)$ and $l(d_2)$, and similarly $r(d)$ lies between $r(d_1)$ and $r(d_2)$. ∎

We sketch now an algorithm to find a monotone path from s to any point t. The algorithm in fact produces the *monotone path map* with respect to s (MPM(s)), which is a subdivision of the plane into sets of points t according to a common structure of the interval of directions along which there is a monotone path from s to t. Figure 4.3 gives an example of such a subdivision. The subdivision itself may have quadratic size, but we will show how to represent it implicitly as two linear-size subdivisions such that for any query point t we can quickly determine the interval of monotone directions from s to t.

The algorithm proceeds by simulating the sweeping of the paths $l(d)$ and $r(d)$ as d revolves counterclockwise about s. At each critical direction d for which either $l(d)$ or $r(d)$ changes discontinuously, we process an "event". Events for path $l(d)$ are of two types: Type I, in which a segment along $l(r)$ encounters a left point of tangency; and Type II, in which a segment along $l(r)$ encounters a right point of tangency. Refer to Figure 4.4. (Analogous events are defined for path $r(d)$.) Note that an initial pair of paths $l(d)$ and $r(d)$ can be found in time $O(n \log n)$ by a sweep line method (using a sweep line orthogonal to d), or can be read off of the VAG(d) in linear time. Note also that the events will correspond to directions of common tangency lines between pairs of obstacles, and these directions are easily obtained from the visibility graph.

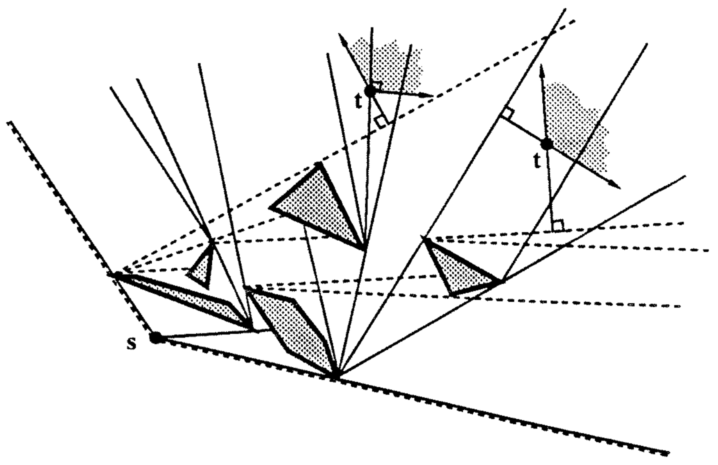

Figure 4.3. Example of a monotone path map.

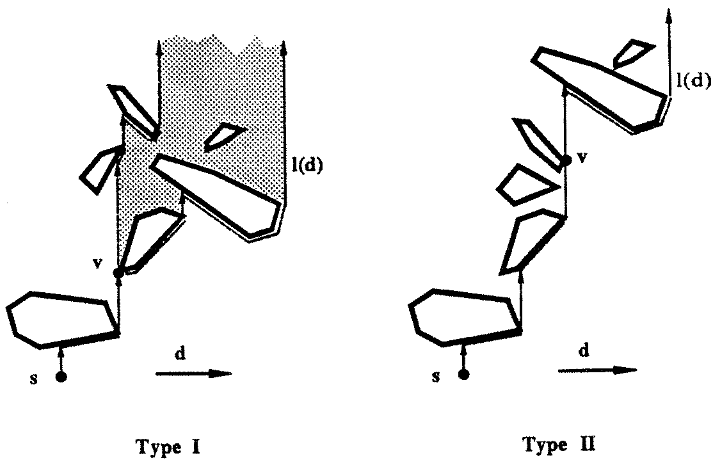

Figure 4.4. Events for sweeping $l(d)$.

The algorithm maintains a description of the current paths $l(d)$ and $r(d)$, and at each event makes the appropriate changes to one of them. Type II events cause a simple insertion to be made into $l(d)$. Type I events, though, may modify a large portion of the path $l(d)$, as seen in the figure. But, the region between the old and the new path $l(d)$ will never be visited again by the sweeping of $l(d)$ (by Lemma 4.2), so vertices that define its boundary are charged only once. The result is the following.

Lemma 4.9 *There will be at most $O(k)$ events.*

Corollary 4.10 *The monotone path map can be represented implicitly in a data structure of size $O(n)$, such that in $O(\log n)$ query time we can retrieve the* *cone of all possible directions along which there is a monotone path from s to a query point t.*

In Figure 4.3, we have shown two values of query point t, and we have shaded the corresponding cones of directions d for which there exists a monotone path from s to t. The representation of the MPM is simply two planar subdivisions (each of size $O(n)$), one corresponding to the critical events in sweeping the path $l(d)$ (shown in solid lines in the figure), and one corresponding to the critical events in sweeping $r(d)$ (shown in dashed lines in the figure).

In order to complete the sketch of the algorithm, we must describe how we determine the next event. For each vertex v along the path $l(d)$ (and similarly for vertices along $r(d)$), we assume that we have all other vertices visible from v in sorted order by angle about v. This can be done in an $O(E+n\log n)$ preprocessing step by constructing the visibility graph, since the algorithm of [GM] actually produces, for each vertex, the edges of the visibility graph sorted by angle about that vertex. (Fortunately, we will not need to have the complete list of visibility graph edges sorted, as this would require $O(E\log n)$ using present technology.) Thus, for each of the $O(n)$ vertices along $l(d)$ and $r(d)$, we know when the next event will occur. We keep these potential upcoming events stored in a priority queue. The next event will be the one stored at the head of the queue. Theorem 4.8 implies that we need only $O(n)$ insertions/deletions in the queue. The result is the following.

Theorem 4.11 *The monotone path map can be constructed in time $O(n\log n)$ for any particular source point s, after having completed an $O(E+n\log n)$ preprocessing.*

Remark: In the full paper, we go on to show how monotone path maps can be constructed for *nonconvex* obstacles as well. We also introduce the notion of a *shortest monotone path map* subdivision, which allows us not only to find a monotone path from s to any other point t, but allows us to find the *shortest* such path. We analyze the combinatorial complexities of both monotone path maps and shortest monotone path maps.

5. Separation Problems

The problem of detecting "movable separability" of sets has received much attention in the recent literature (see [To] for a survey of many recent results). One class of problems that has been addressed is that of determining when a given pair of objects can be separated by a single translation. We can cast this question into the context of monotone paths by the following result.

Lemma 5.1 *A set S of objects in the plane can be separated into two disjoint sets S_1 and S_2 ($S = S_1 \cup S_2$) by a motion of objects S_1 in direction d if and only if there exists a monotone path in direction $d_\perp$ between S_1 and S_2.*

Proof: This can be seen as a consequence of Theorem 7 of [To]. ▮

We can apply our results on monotone paths to solve the following separation problem: Given a set S of polygonal parts in the plane, find a direction d and a subset $S_1 \subset S$ ($S_1 \neq \emptyset, S$) such that the set of parts S_1 can be separated from the parts $S_2 = S \setminus S_1$ by a single translation of S_1 in direction d. If such a separation of S is possible, we say that S can be *fractured* with respect to direction d.

We sketch our method here. First, find the convex hull of the set S. Next, for each segment $\overline{p_i q_i}$ of the convex hull boundary which joins a vertex p_i of one object to a vertex q_i of a different object, we construct a pair of very long line segments attached at points p_i and q_i which extend out from S in a direction perpendicular to $\overline{p_i q_i}$. (The extensions need only be long enough that their convex hull strictly includes S. Their purpose is to avoid our algorithm producing the trivial monotone path that follows the boundary of the convex hull.) Refer to Figure 5.1. Let Ω be the space whose obstacle set consists of S appended with the segments just described.

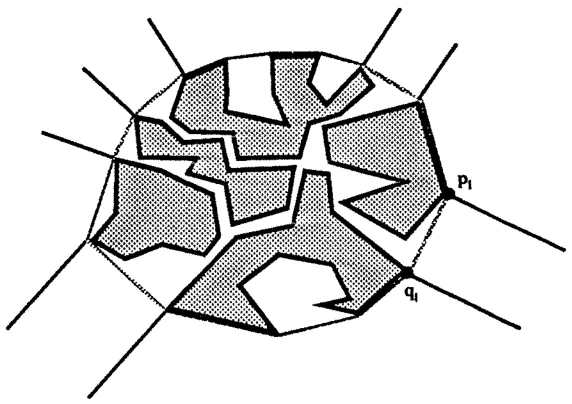

Figure 5.1. Construction for fracturing a set of parts.

Lemma 5.2 *There exists a direction d such that S can be fractured with respect to direction d if and only if there exists a monotone path through Ω between a point of segment $\overline{p_iq_i}$ and a point of segment $\overline{p_jq_j}$, for $i \neq j$. In fact, it suffices to check only for paths between endpoints of the segments $\overline{p_iq_i}$ and $\overline{p_jq_j}$.*

Lemma 5.2 provides the basis for an algorithm. We can solve the $O(n^2)$ monotone path problems by techniques described in Section 3. We stop as soon as we find some monotone path. We then partition S accordingly, and then we can reapply the same technique to each subset, each time separating at least one part from the remainder. This gives an algorithm for finding a sequence of motions to assemble a set of parts, if such a sequence exists. The naive means of implementing this algorithm would run in time $O(n^2 \cdot nE \cdot n) = O(n^4 E)$. We give instead a means of achieving $O(nkE)$ by searching simultaneously for all $O(n^2)$ pairs of monotone paths. Figure 5.2(a-c) shows an example of our algorithm finding a sequence of fracturings that leads to the disassembly of a set of parts.

Theorem 5.3 *The problem of planning an assembly of parts under the restrictions described above (single translational separations) can be solved in time $O(nkE)$ and space $O(E)$.*

Proof: (Sketch) The main idea of our technique is to perform simultaneously all of the steps necessary for computing monotone paths between convex hull points of all sets of parts that remain to be disassembled. We thereby will spend $O(nE)$ per level in the disassembly. Since there are at most k levels, the result will follow.

We modify our $O(nE)$ algorithm of Section 3 as follows. For each of the E critical directions d_e, we build the DVAG(d_e). But now we also identify the important set V of vertices which are on the convex hull(s) of the assemblies we need to take apart. (We are solving many disassembly problems at once. For each, we have found the convex hull and attached at most $O(n)$ extension segments.) For each node v of the DVAG which corresponds to a vertex of V, we "split" the node in two (v_I and v_O), with v_I serving as the "head" for all incoming arcs to v and v_O serving as the "tail" for all outgoing arcs from v. We draw an arc from a "super source" σ to each v_O, and an arc from each v_I to a "super sink" τ. We then search for a path from σ to τ in the resulting digraph. This search takes time $O(n)$ per direction E, for a total of $O(nE)$ per level of fracturing. ∎

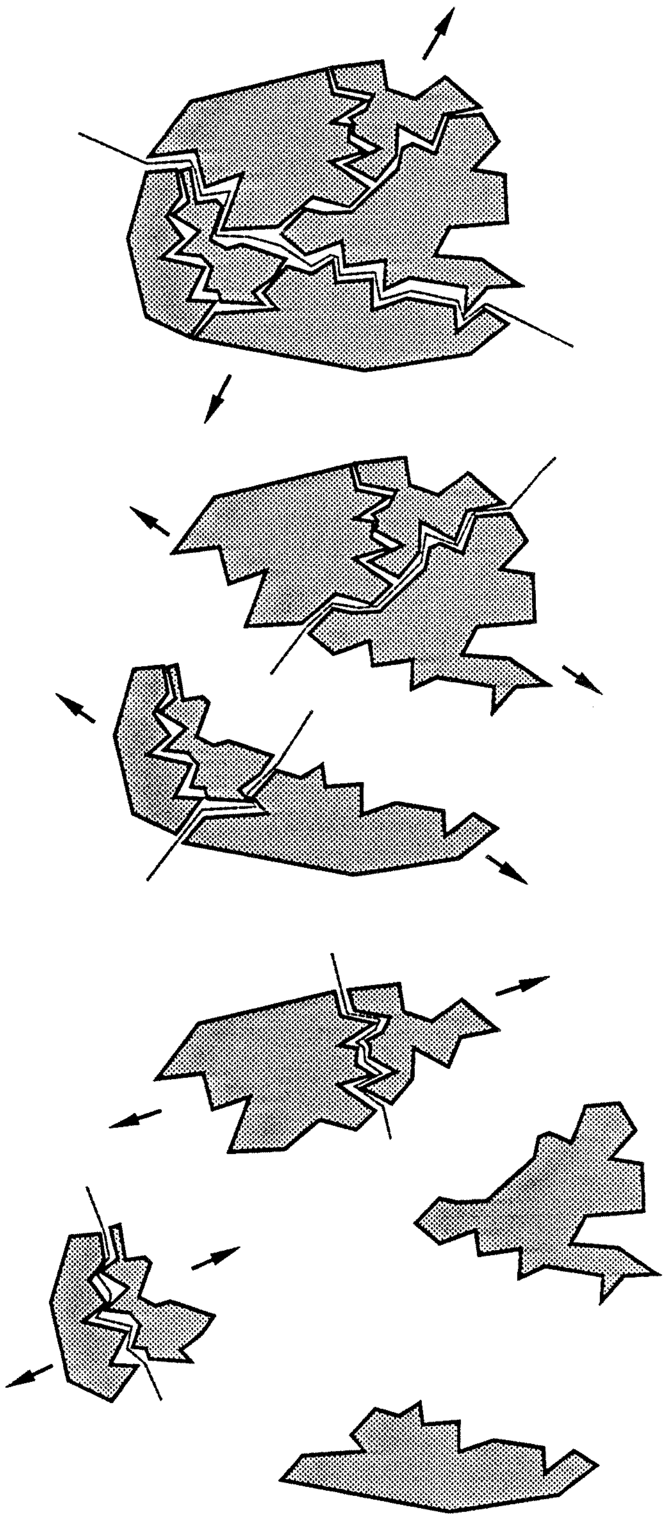
Figure 5.2(c). A disassembly sequence of fracturings.

Can we get a disassembly sequence which is close to "balanced", that is, in which the set of parts is (roughly) divided in half at each separation? The answer is that we can in the case of convex parts (we do so by applying the results of [GY]), but in general it will not be possible to get a balanced disassembly sequence. Figure 5.3 shows a case in which the disassembly requires k separation steps. An interesting open question is whether or not we can find a minimum depth disassembly sequence.

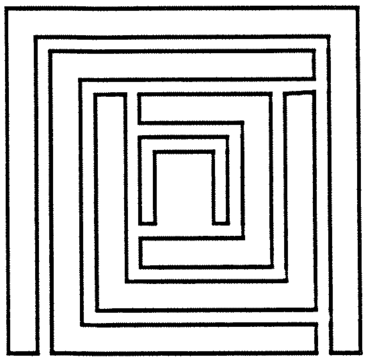

Figure 5.3. Depth k disassembly required.

6. Conclusion

Several extensions of our results are possible, including the following.

1). Suppose that instead of requiring a path to be monotone in direction d, we require every subvector of the path to lie within some given cone. (Call this monotonicity with respect to the cone.) Monotonicity in direction d is the special case in which the cone is the halfspace defined by d. Alternatively, we may not be given a fixed cone but rather only the angle subtended by the cone. The problem then is to determine if there exists an orientation of the cone such that there is a path that is cone-monotone. Our results generalize to handle these cases.

2). We may want to search for a path which "switches back" with respect to d the minimum number of times (i.e., which is the least non-monotone in direction d, in this sense). This can be solved by a simple modification of our algorithm that minimizes travel in a fixed direction.

Some interesting open problems are suggested by our research. We are currently examining the following questions.

1). How can we characterize and detect the case in which it is possible to separate 2 or more polygons by a sequence of at most 2 translations? The existence of a monotone path between a set of objects is a proof that they can be separated by a single translation. But if we allow up to 2 (or more generally, up to k) translations, it is more difficult to provide a succinct proof of separability. (Standard motion planning methods can, of course, be applied.)

2). What about higher dimensions? We are quite interested in the problem of computing a monotone *surface* among a set of polyhedral obstacles in three dimensions. Such surfaces could be used to plan assemblies of three-dimensional parts. Our results on this problem are only preliminary at this time.

Acknowledgement

We would like to thank Robert Freimer, Samir Khuller, Mark Novick, Shmuel Onn, Christine Piatko, and all of the other participants that have been attending the Computational Geometry Workshops at Cornell, where the problems addressed here were originally raised.

7. References

[CR] J. Canny and J. Reif, "New Lower Bound Techniques for Robot Motion Planning Problems", *Proc. 28th FOCS*, pp. 49-60, Oct. 1987.

[GM] S.K. Ghosh and D.M. Mount, "An Output Sensitive Algorithm for Computing Visibility Graphs", Technical Report CS-TR-1874, Department of Computer Science, University of Maryland, July 1987. (Also appears in FOCS, 1987.)

[GH] L.J. Guibas and J. Hershberger, "Optimal Shortest Path Queries in a Simple Polygon", *Proc. Third Annual ACM Conference on Computational Geometry*, Waterloo, Ontario, 1987, pp. 50-63.

[GY] L.J. Guibas and F.F. Yao, "On translating a set of rectangles", *Proc. 12th Annual ACM Symposium on Theory of Computation* (1980) 154-160.

[PS] F.P. Preparata and K.J. Supowit, "Testing a Simple Polygon for Monotonicity", *Information Processing Letters* **12**, No. 4 (1981), pp. 161-164.

[To] G.T. Toussaint, "Movable Separability of Sets", in *Computational Geometry*, Ed. G.T. Toussaint, North Holland Publishing Co., 1985.