Information Processing Letters 20 (1985) 259–263
North-Holland

12 June 1985

AN ALGORITHM FOR SHORTEST-PATH MOTION IN THREE DIMENSIONS *

Christos H. PAPADIMITRIOU **

Department of Computer Science, Stanford University, Stanford, CA 94305, U.S.A.

Communicated by M.A. Harrison
Received 11 June 1984

We describe a fully polynomial approximation scheme for the problem of finding the shortest distance between two points in three-dimensional space in the presence of polyhedral obstacles. The fastest algorithm known for the exact solution of this problem is doubly exponential.

Keywords: Shortest path, path planning, polyhedral obstacle

1. Introduction

Given two points and several polyhedral obstacles in three-dimensional space, we wish to find the shortest path between the given points that avoids the obstacles. This is an interesting algorithmic problem motivated by motion planning, recently posed by Sharir and Schorr [7]. The two-dimensional case with polygonal obstacles is a rather well looked at problem; the fastest algorithm known for it is $O(n^2 \log n)$ [7], and $O(mn \log m)$, where m is the number of polygonal obstacles, when these obstacles are convex [5]. The special case of the three-dimensional problem in which the two endpoints lie on the surface of a convex polyhedron (and thus the shortest path is going to be a sort of *geodesic* of the polyhedron) was also solved in polynomial time in [7]. However, the general three-dimensional problem appears to be much harder. The difficulty seems to come from the complex nonlinear metric imposed by two skew lines in three-dimensional space. For example, there is no efficient algorithm known for finding the shortest path touching several such lines in a specified order. The obvious algebraic formulation of the problem gives rise to a system of simultaneous quartic equations, which take doubly exponential time to solve [7].

In this paper we give a *fully polynomial approximation scheme* for the general three-dimensional shortest-path problem. That is, given an instance of the problem and a positive real ε, we find a path which is of length at most $(1 + \varepsilon)$ times the length of the shortest path, in time that is polynomial in the size of the instance *and* $1/\varepsilon$. The basic idea in our algorithm is quite straightforward: Since it is so hard to compute shortest paths through skew lines, we break such lines into a number of short segments. These segments are such that the distance between two points on two segments can be considered constant and independent of the precise position of two points within the two segments, and the resulting relative error in the calculation of shortest distances does not exceed ε. We then solve a shortest path problem in a graph with the segments as nodes. It turns out that this breaking must be done in a particular nontrivial way, if the relative error is to remain bounded by ε and the number of segments polynomial.

In the next section we present the basic ideas, which lead to an algorithm with complexity

* This research has been supported by a grant from the National Science Foundation, by the Hughes Artificial Intelligence Laboratory, and by the Hellenic Ministry for Research and Technology.

** Present affiliation: National Technical University of Athens, 157 73 Athens, Greece.

$O(n^4(L + \log(n/\varepsilon))^2/\varepsilon^2)$; here, n is the number of elements (vertices, edges, and faces) of the polyhedral scene, ε the desired accuracy of the approximation algorithm, and L the precision of the integers used, that is, the number of bits in the largest integer describing the coordinates of any scene element. (*Note*: It is rather unfortunate, but seemingly necessary, that L must appear in our estimates of the efficiency of our algorithm, as it does, for example, in Khacian's famous algorithm for linear programming [3]. Such algorithms are not *strongly polynomial* in the terminology of [6].) To better understand this bound, notice that the term $K = L + \log(n/\varepsilon)$ is essentially the *precision* of the instance, as it contains the logarithm of n plus the logarithm of the largest integer used in the instance, plus the number of leading zero bits of ε.

In Section 3 we discuss a possible improvement of the running time to $O(n^3K^2/\varepsilon)$, other evidence about the complexity of this problem, and we argue about the practicality of our approach.

2. The algorithm

We are given a *polyhedral scene*, that is, a set of polyhedra given in terms of the coordinates of their vertices, the endpoints of their edges, and the perimeters of their faces, and two points s and t. We assume that s does not lie on any edge of the scene. This is not a loss of generality. There are n vertices in the scene, and thus O(n) edges and faces. We assume that the coordinates of all points are given as integers. Let L be the logarithm (base 2) of the largest integer appearing as a coordinate. Both n and L characterize the size of the instance in hand.

We wish to find the shortest path from s to t; that is, the shortest continuous curve connecting s and t that avoids all polyhedra in the scene. By 'avoiding' we mean that the curve is disjoint from the union of the open polyhedra. It is easy to see that the shortest curve will be a broken straight line, with all breakpoints lying on edges of the polyhedra [7].

Consider an edge ζ of the scene, and the distance a from point s to ζ. We define a coordinate system on ζ with the origin at the closest point to s, and a sequence of points on ζ with coordinates $x_0 = 0$, $x_j = \varepsilon_1 a(1 + \varepsilon_1)^{j-1}$, $j = 1, 2, \ldots$, and their negatives $x_{-j} = -x_j$. ε_1 is a small constant to be fixed later. We have thus divided the edge into several segments. We subdivide this way all lines in our scene. Naturally, we stop at the ends (i.e., vertices) of the edges of the scene. How many such segments are there totally?

Lemma 1. *There are at most* $N = O(n(L + \log(1/\varepsilon_1))/\varepsilon_1)$ *segments.*

Proof. The longest edge of our scene is not longer than 2^{L+1}. The shortest distance between any pair of nonintersecting lines cannot be smaller than 2^{-3L} (remember, this distance is the distance of two parallel planes with integer coefficients with at most L bits, and thus a 3×3 determinant of such integers). The sequence of points defined on each line is a geometric progression with ratio $1 + \varepsilon_1$. Taking logarithms, we conclude that each line will be subdivided into $O((L + \log(1/\varepsilon_1))/\varepsilon_1)$ segments. □

We denote the maximum number of segments in any line by M. These segments are the basic elements of our algorithm. Before calculating the distances between any two segments, we must first determine, for each pair of segments, that there are two points, one on each, which are 'visible' from the other. If two such points exist, we say that the two segments are visible from each other. The way this relation can be computed is explained in the following lemma.

Lemma 2. *We can construct the visibility relation of the segments in* $O(N^2 + n^4 \log M + Mn^3 \log n)$ *time.*

Proof. This problem is quite intricate, despite its deceivingly simple statement. Our method is based on ideas reported in [2], although this proof is reasonably self-contained.

We fix a pair of lines, and we compute the visibility relation of the segments on these two lines. Repeating for each pair, we obtain the whole relation. Let us fix a coordinate system (x for one line, y on the other), and let us study the region R

of the (x–y)-plane which represents pairs of points of the two lines visible from one another. We first observe that two points x and y on these lines are not visible from each other (that is, $(x, y) \notin R$) if and only if their line intersects one of O(n) triangles in which we can subdivide the surfaces of the polyhedra of our scene. Let T_i be one such triangle, and let R_i be all x–y pairs such that the line between the two corresponding points does not intersect triangle T_i. Thus, $R = \cap R_i$. We shall next show that the R_i's have a particularly nice structure.

The line defined by the two points with coordinates x and y in the perspective systems intersects the triangle if it intersects the three half-planes defining the triangle. Now, it is an interesting elementary observation that the necessary and sufficient condition for such a line to intersect a half-plane is of the form $(x - a)(y - b) < c$ or $(x - a)(y - b) > c$ [2]. (*Proof*: We shall show that the points x and y on the two lines which also lie at a common plane through a fixed third line ζ satisfy $(x - a)(y - b) = c$ for appropriate a, b, and c. Consider a system of cylindrical coordinates with axis the line ζ. If θ is the angle of the plane, then we must have, for appropriate constants $a_1, \ldots, b_4$ defining the two lines

$$\tan\theta = \frac{a_1x + a_2}{a_3x + a_4} = \frac{b_1y + b_2}{b_3y + b_4},$$

from which we obtain $(x - a)(y - b) = c$.)

It follows that each region R_i is the intersection of three hyperbolic regions, which is easy to construct in constant time. By *construct* we mean to determine its vertices—i.e., points belonging to more than one hyperbolic arc of the boundary —and the equations of the hyperbolic arcs between any two adjacent vertices. We must now form the intersection of O(n) such regions. We can do this by the obvious incremental algorithm in $O(n^2)$ time. The output of this process is a planar graph of degree two (the boundary of R), with points of the (x–y)-plane as vertices, and hyperbolic arcs as edges. Up to now, the time is $O(n^2)$ for this pair of lines.

Having done this, we can compute the visibility relation of the segments of the two lines as follows. We essentially wish to know, for each of the $O(M^2)$ rectangles in which the (x–y)-plane is divided by the two subdivisions of the axes into O(M) segments, whether the rectangle intersects R. This is done as follows: We traverse the boundary of R, and, for each vertex we determine the rectangle in which it belongs (this takes time O(log M) for each vertex, and $O(n^2 \log M)$ in all). For each arc, we find the boundaries of rectangles it intersects (time O(Mn), since we have at most one intersection of each hyperbola with each straight line). By sorting these intersection points on each horizontal or vertical line (time O(Mn log n)) we can compute whether each rectangle intersects with R: We just check whether its boundary intersects the boundary of R or else is in the interior of R, and if not, whether it contains a vertex of the boundary of R. If all fail, the rectangle has no intersection with R, and the corresponding two segments are totally invisible from each other.

The total time, for a pair of lines, comes to $O(n^2 \log M + Mn \log n + M^2)$. For the whole scene, the bound is $O(N^2 + n^4 \log M + Mn^3 \log n)$, which is the bound to be proved. □

We then compute the distance of any two segments related by the visibility relation. Their distance is the distance of the midpoints of the segments. Segments that are not visible from one another have infinite distance. The way that the segments were defined guarantees that the error made by considering the distance between any two points of two segments as constant is not too great. In particular, we can prove the following lemma.

Lemma 3. *Let* v *be a segment,* $\ell(v)$ *its length, and* D(v) *its minimum straight-line distance from* s. *Then* $\ell(v) < \varepsilon_1 D(v)$.

Proof. Suppose that v is the segment $[x_j, x_{j+1}]$, $j \geq 0$. Let a be the distance from s to the line of v. Then $D(v) \geq \max\{a, x_j\}$, and, by the definition of the x_j's, $\ell(v) \leq \varepsilon_1 \min\{a, x_j\}$. □

We then apply Dijkstra's algorithm [1,6] in the resulting graph. If the shortest path found has infinite cost, then there is no obstacle-avoiding

path in our scene. Otherwise, the shortest path from s to t found by the algorithm is transformed into an actual three-dimensional obstacle-avoiding path as follows: We replace the edge from segment v to segment w by any obstacle-avoiding segment joining any point of v with any point of w (since the distance between v and w is finite, such a segment exists). We connect these endpoints by straight lines. Suppose that the path yielded by our algorithm is $Q = s, q_1, r_1, \ldots, q_k, r_k, t$, where q_j and r_j are points of the same segment w_j, and let $P = s, p_1, \ldots, p_\ell, t$ be the actual shortest path, with point p_j lying on segment v_j. We let $d(x, y)$ stand for the distance between the points x and y. If P is a path, then $d(P)$ is its length.

Lemma 4. *Suppose that* $\varepsilon_1 \leqslant 1/8n$. *Then the path yielded by our algorithm satisfies* $d(Q) \leqslant (1 + 4\varepsilon_1 n)d(P)$.

Proof. Since Q is the result of the algorithm, we have that

$$d(s, m(w_1), \ldots, m(w_k), t) \leqslant d(s, m(v_1), \ldots, m(v_\ell), t),$$

where by $m(v)$ we denote the midpoint of the segment v. Also,

$$d(Q) \leqslant d(s, m(w_1), \ldots, m(w_k), t) + \sum_{j=1}^{k} \ell(w_j)$$

and

$$d(s, m(v_1), \ldots, m(v_\ell), t) \leqslant d(P) + \sum_{j=1}^{\ell} \ell(v_j),$$

from the triangle inequality in Euclidean 3-space. It follows from the above that $d(Q) \leqslant d(P) + \sum_{j=1}^{k} \ell(w_j) + \sum_{j=1}^{\ell} \ell(v_j)$. However, recall that, by Lemma 3, $\ell(v) \leqslant \varepsilon_1 D(v)$. Also, it is easy to see that $D(v_j) \leqslant d(P)$, and $D(w_j) \leqslant d(Q)$. Finally, observe that $\ell \leqslant n$, because no two segments of the same edge are in the optimal path from s to t; as for the path yielded by our algorithm, it is easy to see that $k \leqslant 2n$, because at most two segments from each edge can participate. This implies that

$$d(Q)(1 - 2n\varepsilon_1) \leqslant d(P)(1 + n\varepsilon_1).$$

Using the given bound for ε_1, the lemma follows. □

Theorem. *There is an algorithm which, given a scene with* n *edges and coordinates with precision* L, *and an* $\varepsilon > 0$, *yields a path with relative error at most* ε *in* $O(n^4(L + \log(n/\varepsilon))^2/\varepsilon^2)$ *time.*

Proof. Just take $\varepsilon_1 = \varepsilon/4n$, recalling Lemmas 1, 4, and the fact that Dijkstra's algorithm works in $O(N^2)$ time. □

3. Discussion

There is one, rather complicated for the result achieved, way to improve this by a factor of n/ε. The idea is this: We do not assume the distance between two points on the same segment to be constant, but a *linear function* of the coordinates. That is, suppose that we have two segments with midpoints x_1 and y_1, where we have now taken as the origin of the coordinate systems on the two lines the points of smallest distance of the two lines. Instead of taking the distance between any two points x and y of these segments to be, say, $d(x_1, y_1)$, we can take it to be

$$d(x_1, y_1) + (x - x_1)\frac{x_1 - y_1 \cos\theta}{d(x_1, y_1)} + (y - y_1)\frac{y_1 - x_1 \cos\theta}{d(x_1, y_1)},$$

where θ is the angle of the two lines. This is the Taylor expansion of the distance around (x_1, y_1), truncated after the linear term. Our shortest path problem now becomes a continuous version in which for each node v we can choose a parameter $\lambda(v) \in [0, 1]$, and the distance between u and v is now of the form $a + b\lambda(u) + c\lambda(v)$. We must choose the optimum sequence of nodes s, $v_1, \ldots, v_k$, t *and* parameters $\lambda(v_1), \ldots, \lambda(v_k)$ such that the total distance is minimum.

Fortunately, we can solve this harder version of the shortest path problem in $O(N^2)$ time as well. Just observe that the optimum has values for the λ's that are either 1 or 0. As a consequence, the problem reduces to an ordinary shortest path

problem with 2N nodes. What we have achieved is that the error in approximating the distances is now smaller. Each leg of the path introduces an error of $O(\varepsilon_1^2)$, and thus in the proof of the Theorem we could take $\varepsilon_1 = \sqrt{\varepsilon/n}$, for a bound of $O(n^3K^2/\varepsilon + n^4K)$. We can get rid of the second term by relying on a visibility relation in which two segments are related if *all* their points are visible. The ideas is that, if two segments are partially obscured by a polyhedron, each sees all of a segment of the obscuring polyhedron, or, if not, of the polyhedron which partially obscures them, and so on. The details are rather unpleasant, since the counterpart of Lemma 4 would now require an argument for handling segments that are not obstacle-free (this argument is similar to the one for partial visibility outlined above).

Is the three-dimensional shortest-path problem NP-complete? For one, it does not seem to be in NP at all, because of the difficulty with the skew lines mentioned before. This difficulty is reminiscent of a similar one with the Euclidean traveling salesman problem and the precision required in evaluating tours [6], although the present situation is far more complex, and, furthermore, there is no obvious discretization to help avoid the issue. If we are tempted to do some kind of discretization (e.g., allow breakpoints of the path with only integer coordinates) we conjecture that the resulting problem is NP-hard, although such a variant would lack most of the complexities of the original problem. Naturally, our algorithm suggests that it is not *strongly* NP-hard.

It should be mentioned that one desirable feature of our approach is that its main algorithmic part is a shortest path computation. Computing shortest paths is among the problems that can be *parallelized* easiest. By this we mean that the speed-up obtained by using many processors on the problem is very close to the number of processors used, and the total delay can be brought down to the same power of the *logarithm* of the input size. We can find shortest paths in parallel by adopting the matrix multiplication version of the Bellman–Ford algorithm [4], and parallelizing the matrix computation in the obvious way. Finally, the visibility computation can be naturally parallelized. The class of problems that are so nicely parallelizable (with parallel time which is polynomial in the *logarithm* of the input, and a polynomial number of processors involved) is usually denoted NC. Our algorithm meets these strict criteria, since the preprocessing step can be easily parallelized, and the rest is a shortest path computation.

Our algorithm is probably not fast enough to be immediately practical, but it is not of 'strictly theoretical interest' either. For a scene with 50 lines and three decimal places of precision, we can get solutions that are at most 10% of the optimum in something like 10^{10} steps. Naturally, this is far better than the doubly exponential exact (or even the exponential exhaustive approximate) approach suggested in [7]. Although we do not know how to improve the asymptotic time requirements of our algorithm below n^3K^2/ε, it is quite apparent that both the algorithm and the argument are rather wasteful by accounting for worst cases, both in terms of the time requirements and the error of the solution found. We conjecture that our result can be the basis of a reasonable approach to this problem. In fact, we would not be very surprised if this idea is already in use in some existing motion planning systems.

Acknowledgment

Many thanks are due to Rod Brooks for his criticism of my original algorithm for visibility, which led to Lemma 2, and to Leo Guibas for helpful discussion concerning this lemma.

References

[1] E.W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 259–261.

[2] L. Guibas and C.H. Papadimitriou, Visibility and intersections of hyperbolic regions, In preparation, 1984.

[3] L.G. Khacian, A polynomial algorithm for linear programming, Dokl. Soviet Acad. Sci., 1979.

[4] E.L. Lawler, Combinatorial Optimization: Networks and Matroids (Holt, Rinehart & Winston, New York, 1976).

[5] J. Mitchell, D. Mount and C.H. Papadimitriou, The discrete geodesic problem, Unpublished manuscript, 1984.

[6] C.H. Papadimitriou and Steiglitz, Combinatorial Optimization: Algorithms and Complexity (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[7] M. Sharir and A. Schorr, Proc. 16th STOC (1984) 144–153.