Purdue University

Purdue e-Pubs

Department of Computer Science Technical Reports

Department of Computer Science

1992

On Parallel Rectilinear Obstacle-Avoiding Paths

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Danny Z. Chen

Report Number:

92-045

ON PARALLEL RECTILINEAR OBSTACLE-AVOIDING PATHS

Mikhail J. Atallah
Danny Z. Chen

CSD-TR-92-045
August 1992

On Parallel Rectilinear Obstacle-Avoiding Paths*

Mikhail J. Atallah† Danny Z. Chen‡

Abstract

We give improved space and processor complexities for the problem of computing, in parallel, a data structure that supports queries about shortest rectilinear obstacle-avoiding paths in the plane, where the obstacles are disjoint rectangles. That is, a query specifies any source and destination in the plane, and the data structure enables efficient processing of the query. We now can build the data structure with $O(n^2/\log n)$ CREW PRAM processors, as opposed to the previous $O(n^2)$, and with $O(n^2)$ space, as opposed to the previous $O(n^2(\log n)^2)$. The time complexity remains unchanged, at $O((\log n)^2)$. As before, the data structure we compute enables a query to be processed in $O(\log n)$ time, by one processor for obtaining a path length, or by $O(\lceil k/\log n \rceil)$ processors for retrieving a shortest path itself, where k is the number of segments on that path. The new ideas that made our improvement possible include a new partitioning scheme of the recursion tree, which is used to schedule the computations performed on that tree. Since a number of other related shortest paths problems are solved using this technique as a subroutine, our improvement translates into a similar improvement in the complexities of these problems as well.

1 Introduction

Let P be a rectilinear convex polygon having $O(n)$ vertices and inside which lie n pairwise disjoint rectangular obstacles that are rectilinear (i.e., whose edges are parallel to the coordinate axes). We are interested in computing, in parallel, a data structure that supports queries about shortest rectilinear obstacle-avoiding paths in P. That is, a query specifies a source and a destination, and the data structure enables efficient processing of the query. For any pair of query points, the data structure computed in [3] enables, in $O(\log n)$ time, one processor to obtain the path length, or $O(\lceil k/\log n \rceil)$ processors to retrieve the shortest path itself, where k is the number of segments of that path. Here we construct the same data structure as in [3], but by using $O(n^2/\log n)$ CREW PRAM processors rather than $O(n^2)$, and with $O(n^2)$ space rather than $O(n^2(\log n)^2)$. The time complexity of the algorithm for constructing the data structure remains $O((\log n)^2)$. The new ideas that made

*This research was supported by the Leonardo Fibonacci Institute in Trento, Italy. Additional support was provided by the Air Force Office of Scientific Research under Grant AFOSR-90-0107, and by the National Library of Medicine under Grant R01-LM05118.

†Dept. of Computer Science, Purdue University, West Lafayette, IN 47907. Part of this research was done while this author was visiting LIPN, Paris, France.

‡Dept. of Computer Science, Purdue University, West Lafayette, IN 47907.

our improvement possible include a new partitioning scheme of the recursion tree, and the careful use of this partitioning to schedule the computations performed on T. This results in a smaller processor complexity, and also in a saving in space made possible by the fact that we can now throw away information almost immediately after using it (whereas the scheme in [3] was forced to keep that information).

We refer the reader to [3] for a more detailed discussion of such path problems and for a review of the previous work on such problems. The next section reviews the definitions and features of the algorithm in [3] that are needed to comprehend the improvement which will be given later, in Section 3.

Recall that the CREW PRAM is the synchronous shared-memory model where concurrent reads are allowed, but no two processors can simultaneously attempt to write in the same memory location (even when they are trying to write the same thing).

Throughout, we assume that all geometric objects (segments, polygons, paths, rectangles, convex hulls, etc.) are rectilinear (that is, each of their constituent segments is parallel to one of the two coordinate axes), and that all paths (shortest or otherwise) are obstacle-avoiding.

2 A Review of the Previous Algorithm

Polygon P is specified by a circular sequence of vertices $v_1, v_2, \ldots, v_m$, as encountered by a counterclockwise walk along the boundary of P starting at v_1, where m is the number of vertices of P. A circular ordering of the points on the boundary of P is defined by the order in which they are encountered in the walk along the boundary of P that follows the circular sequence of vertices of P. The set of rectangular obstacles contained in P is denoted by R. The vertex set of R is denoted by V_R (hence $|V_R| = 4n$).

A rectilinear convex polygon Q is a rectilinear simple polygon such that every line segment which joins two points of Q and is parallel to a coordinate axis is contained in Q. The rectilinear convex hull of a set of objects in the plane is a (rectilinear) convex polygon that contains the set of objects and has minimum area.

For a set of obstacles S, it is possible that the convex hull of S does not exist (see [12] for example). Let $CH(S)$ denote the convex hull of S (if $CH(S)$ exists). Let R' be a subset of R, and without loss of generality assume that $CH(R')$ exists (Section 2 of [3] shows how to handle the case where $CH(R')$ does not exist). Furthermore, we assume that $CH(R')$ does

Figure 1: Illustrating the definition of $B(R')$.

not intersect the interior of any obstacle in $R - R'$ (the way in which the algorithm in [3] partitions obstacles into subsets guarantees that this assumption holds for all the subsets of R generated by the algorithm). In the following definition, when we talk about "visibility", we are assuming that the obstacles as well as $CH(R')$ are opaque objects.

Definition 1 *Let $B(R')$ be the set of points p on $CH(R')$ such that either (i) p is a vertex of $CH(R')$ or (ii) p is horizontally or vertically visible from a vertex in $V_{R'}$ (see Figure 1).*

Obviously, $|B(R')| = O(|R'|)$. That $B(R')$ can be computed in $O(\log n)$ time using $O(n)$ processors follows from [4]. We assume that $B(R')$ is sorted according to the order in which its points are visited by a counterclockwise walk around $CH(R')$, starting at some vertex of $CH(R')$. The next lemma shows the importance of $B(R')$.

Lemma 1 *For a vertex $p \in V_{R'}$ and a point q not in the interior of $CH(R')$, there exists a shortest p-to-q path that goes through a point of $B(R')$.*

Proof. See the proof of Lemma 13 in [3]. □

As in [3], an important method used by the algorithm involves multiplying special kinds of matrices. All matrix multiplications in the algorithm are in the $(\min, +)$ closed semi-ring, i.e., $(M' * M'')(i,j) = \min_k \{M'(i,k) + M''(k,j)\}$. A matrix M is said to be *Monge* [1] iff for any two successive rows $i, i+1$ and columns $j, j+1$, we have $M(i,j) + M(i+1,j+1) \leq M(i,j+1) + M(i+1,j)$. For two point sets A and B in the plane, let matrix M_{AB} contain the lengths of shortest paths between the points in A and the points in B. Now, consider two finite point sets X and Y, each totally ordered in some way (so we can talk about the predecessor and successor of a point in X or in Y), and such that the rows (resp., columns) of the path lengths matrix M_{XY} are as in the ordering for X (resp., Y). Matrix M_{XY} is *Monge* iff for any two successive points p, p' in X and two successive points q, q' in Y, we have $M_{XY}(p,q) + M_{XY}(p',q') \leq M_{XY}(p,q') + M_{XY}(p',q)$. The next lemma characterizes the Monge matrices of path lengths used in the algorithm.

Lemma 2 *Let CP be a convex polygon that contains a subset R' of R and whose boundary does not intersect the interior of any obstacle in R. Let X and Y be finite sets of points on the boundary of CP, such that the portion of that boundary spanned by X is disjoint from that spanned by Y. Then the matrix M_{XY} of path lengths between X and Y is Monge.*

Proof. See Lemma 1 of [3]. □

The following well-known lemma [1, 2] is useful.

Lemma 3 *Assume that matrices M_{XZ} and M_{ZY} are Monge, with $|X| = c_1|Z| \leq c_2|Y|$ for some positive constants c_1 and c_2. Then $M_{XZ} * M_{ZY}$, which is also Monge, can be computed in $O(\log|Z|)$ time and $O(|X||Y|)$ work in the CREW PRAM model.*

The next lemma is also needed in the algorithm.

Lemma 4 *Let X, Y, and Z be finite point sets such that for any $p \in X$ and $q \in Y$, a shortest p-to-q path can be chosen to go through Z, where $|X| \leq \alpha$, $|Y| \leq \beta$, and $|Z| \leq \gamma$, such that $\alpha = c_1\gamma \leq c_2\beta$ for some positive constants c_1 and c_2. Assume that X (resp., Y, Z) can be partitioned into a constant number of subsets X_i, $1 \leq i \leq l_X$ (resp., Y_j, Z_k, $1 \leq j \leq l_Y$, $1 \leq k \leq l_Z$) such that all $M_{X_iZ_k}$ and $M_{Z_kY_j}$ are Monge. Given M_{XZ} and M_{ZY}, the matrix M_{XY} can be computed in $O(\log\gamma)$ time and $O(\alpha\beta)$ work in the CREW PRAM model.*

Proof. See Lemmas 4 and 5 in [3]. □

The algorithm in [3] is based on the two-way divide-and-conquer strategy. For the case of computing the matrix of the $B(R)$-to-$B(R)$ path lengths, the algorithm uses the following divide-and-conquer overall scheme:

(i) It partitions the obstacle set R into two subsets R_1 and R_2 of relatively balanced sizes by using a "staircase" separator (such a staircase separator is computed in [3] in $O(\log n)$ time using $O(n)$ processors).

(ii) It solves recursively the two subproblems in parallel, that is, it computes the matrix of the $B(R_1)$-to-$B(R_1)$ path lengths and the matrix of the $B(R_2)$-to-$B(R_2)$ path lengths.

(iii) It performs $O(1)$ Monge matrix multiplications to obtain, from the output of the two recursive calls (i.e., the matrix of the $B(R_1)$-to-$B(R_1)$ path lengths and the matrix of the $B(R_2)$-to-$B(R_2)$ path lengths), the desired matrix of the $B(R)$-to-$B(R)$ path lengths.

The above procedure obtains the matrix of the $B(R)$-to-$B(R)$ path lengths in $O((\log n)^2)$ time and using $O(n^2/(\log n)^2)$ processors (see Section 5 of [3] for a more detailed description). Now, the algorithm in Section 5 of [3] can actually be used to compute much more information than the matrix of the $B(R)$-to-$B(R)$ path lengths. Stage (iii) of that procedure can also compute the matrix of the $B(R_1)$-to-$B(R)$ path lengths and the matrix of the $B(R_2)$-to-$B(R)$ path lengths within the same complexity bounds as those for computing the matrix of the $B(R)$-to-$B(R)$ path lengths. In addition, that algorithm creates (as in [3]) a recursion tree T, in $O((\log n)^2)$ time and using $O(n^2/(\log n)^2)$ processors. Each node v of T is associated with a subproblem (call it P_v; note that P_v is a subset of R generated by this algorithm) and the information (call it I_v) associated with P_v: Specifically, I_v consists of (1) a description of all the $B(P_v)$-to-$B(P_v)$ path lengths and (2) a description of all the $B(P_v)$-to-$B(P_{parent(v)})$ path lengths (if v is not the root of T). Based on this algorithm, [3] computed the matrix of the V_R-to-V_R path lengths in $O((\log n)^2)$ time using $O(n^2)$ processors. That computation of the V_R-to-V_R path lengths is the most difficult part of the algorithm for building the data structure, and it is in fact this computation that caused the space and processor complexities of [3] to be $O(n^2(\log n)^2)$ and (respectively) $O(n^2)$. The next section describes our new approach to this computation. We do not go into the other aspects of the solution given in [3], since they are not directly relevant to what follows.

3 The Improvement

We assume that we have already executed the algorithm in Section 5 of [3], as reviewed in the previous section, and obtained I_v for each node v in T. We now give a high-level description of our new method for computing the desired matrix of the V_R-to-V_R path lengths. We focus only on the computation of this matrix because, once that matrix is available, the same method as in [3] can be used to obtain, in $O(\log n)$ time and $O(n^2/\log n)$ processors, the description of the $4n$ shortest path trees, each rooted at one of the $4n$ vertices of V_R. It

Figure 2: Illustrating the wavefronts of T.

suffices to give an $O((\log n)^2)$ time, $O(n^2 \log n)$ work, and $O(n^2)$ space algorithm for the computation of the matrix of the V_R-to-V_R path lengths; this would imply the claimed $O(n^2/\log n)$ processor bound because of Brent's theorem [6].

We next describe a partition of the nodes of T that will play an important role in guiding the computations that will later be performed in T. Let the i-th ***wavefront*** in T (denoted as WF_i) be the subset of nodes v in T such that $n \cdot 8^{-i-1} < |P_v| \leq n \cdot 8^{-i}$. Let $0, 1, \ldots, h$ be the indices of the nonempty wavefronts in T (clearly, $h = O(\log n)$). Let $\mathcal{P}$ be any root-to-leaf path in T. The following statements are easy consequences of the definition of wavefronts:

1. $\mathcal{P}$ goes through the wavefronts in sorted order — first through WF_0, then WF_1, etc.

2. The wavefronts form a partition of the nodes of T.

3. The last wavefront, WF_h, contains all the leaves of T.

4. $\mathcal{P} \cap WF_i \leq 16$. This is because if u is a child of w in T, then $|P_w|/8 \leq |P_u| \leq 7|P_w|/8$, and 16 is the smallest integer k such that $(7/8)^k \leq (1/8)$.

5. $\sum_{v \in WF_i} |P_v| \leq 64 \cdot n$. This one follows from the previous one — an obstacle vertex can belong to at most 16 nodes of a wavefront, and since there are $4n$ of them the relationship follows.

Figure 2 illustrates the wavefront concept. Note that the last wavefront contains all the leaves.

Remark: The reader may be wondering why we partition the nodes of T in this way and not in a more "natural" way such as, for example, defining WF_i to be the vertices at the i-th level of T. The reason for partitioning the nodes of T in this way is that it is crucial that the nodes in the same wavefront have the same associated problem size, to within a constant factor of each other (as required by lemmas 3 and 4). A partition by levels would fail to satisfy this requirement because two nodes that are at the same level of T can have very different associated problem sizes, e.g., it could be $O(1)$ for one node and $O(n^\epsilon)$ for another node, $0 < \epsilon < 1$. In other words, we would be unable to use lemmas 3 and 4.

Let M_i, $0 \leq i \leq h$, be the collection of all the $B(P_v)$-to-$B(P_w)$ path lengths information for nodes $v, w \in WF_i$. We compute $M_0, \ldots, M_h$ in that order. We will show that M_0 can be obtained in $O(n^2)$ work and $O(\log n)$ time and that, once we have any M_i, we can obtain M_{i+1} also in $O(n^2)$ work and $O(\log n)$ time. A proof of the previous statement would clearly imply a total $O(h \cdot \log n)$ time bound and $O(h \cdot n^2)$ work bound for the computation of all the M_i's.

The next lemma is a simple but important building block in what will follow later.

Lemma 5 *Let nodes $v, w \in T$ be such that:*

1. $c_1|P_v| \leq |P_w| \leq c_2|P_v|$ *for some positive constants c_1 and c_2.*
2. *The $B(P_{parent(v)})$-to-$B(P_{parent(w)})$ path lengths matrix is already available.*

Then we can compute, in logarithmic time and $O(|P_v| \cdot |P_w|)$ work, the following quantities:

1. *The lengths of the $B(P_v)$-to-$B(P_{parent(w)})$ paths and of the $B(P_w)$-to-$B(P_{parent(v)})$ paths.*
2. *The lengths of the $B(P_v)$-to-$B(P_w)$ paths.*

Proof. See Subsection 6.1 of [3]. □

We now explain how to use Lemma 5 to obtain M_0. We start at the root and proceed down the tree, using Lemma 5 as we go along. We do not enter any node in WF_1 until we are done with WF_0. While processing WF_0, there are actually two types of usages of Lemma 5 that take place, as follows. Suppose we have completed the computation of the $B(P_{parent(v)})$-to-$B(P_{parent(w)})$ path lengths information for $parent(v), parent(w) \in WF_0$. In the case where both v and w are in WF_0, we use the lemma to compute the $B(P_v)$-to-$B(P_w)$ path lengths. In the case where only one of v, w is in WF_0 (suppose $v \in WF_0$, $w \in WF_1$), we use the lemma to compute the $B(P_v)$-to-$B(P_{parent(w)})$ path lengths. In the case where both v and w are in WF_1, nothing is done for the pair v, w until the processing on all the nodes in WF_0 is completed.

Note: The rule "do not start WF_{i+1} until we are done with WF_i" requires synchronization that can easily be done in logarithmic time after each usage of Lemma 5. (There are in fact ways to avoid this synchronization, but since we can afford the obvious logarithmic time synchronization we choose to use it, in order not to unnecessarily clutter the exposition.)

Once done with M_i, we move down and process M_{i+1}, again by repeatedly using Lemma 5.

We claim that the total work done by the above scheme is $O(n^2)$ for the computation of M_0, and also $O(n^2)$ for the computation of any M_{i+1} given M_i. To see this, observe that this work is proportional to:

$$\sum_{v,w \in WF_{i+1}} |P_v| \cdot |P_w| = (\sum_{v \in WF_{i+1}} |P_v|) \cdot (\sum_{w \in WF_{i+1}} |P_w|),$$

which is $O(n^2)$ because:

$$\sum_{v \in WF_{i+1}} |P_v| = O(n).$$

The above analysis also implies an $O(n^2)$ space complexity for the algorithm, because once we are done with processing wavefront WF_{i+1}, we can discard the M_i information, since the computation of M_{i+2} will only need M_{i+1}. We ultimately need only keep M_h, which contains the desired matrix of the V_R-to-V_R path lengths: If p (resp., q) is a point that is a vertex of the rectangular obstacle associated with leaf v (resp., w) of T, then both v and w are in WF_h and hence the shortest p-to-q path length is already available in M_h (by definition, M_h contains the $B(P_v)$-to-$B(P_w)$ path lengths for all $v, w \in WF_h$, and in this case each of P_v and P_w consists of a single rectangle).

4 Conclusion

Although the algorithm given here brings the space complexity down to an optimal $O(n^2)$, the work complexity is still a factor of $\log n$ away from the optimal $O(n^2)$ (recall that the sequential complexity of this problem is $O(n^2)$ time [3]). Whether there is an $O((\log n)^2)$ time, $O(n^2/(\log n)^2)$ processor algorithm for this problem remains an interesting open question.

Acknowledgement. Professor Hosam ElGindy [10] has claimed similar bounds but by

using a very different approach, one that is based on using the Pan-Reif techniques rather than the methods based on Monge matrices that we used.

References

[1] A. Aggarwal and J. Park. "Notes on searching in multidimensional monotone arrays (preliminary version)." *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 497–512.

[2] A. Apostolico, M. J. Atallah, L. L. Larmore, and H. S. McFaddin. "Efficient parallel algorithms for string editing and related problems." *SIAM J. Comput.* 19 (5) (1990), pp. 968–988.

[3] M. J. Atallah and D. Z. Chen. "Parallel rectilinear shortest paths with rectangular obstacles." *Computational Geometry: Theory and Applications*, 1 (2) (1991), pp. 79–113.

[4] M. J. Atallah, R. Cole, and M. Goodrich. "Cascading divide-and-conquer: a technique for designing parallel algorithms." *SIAM J. Comput.* 18 (3) (1989), pp. 499–532.

[5] M. J. Atallah and S. R. Kosaraju. "An efficient parallel algorithm for the row minima of a totally monotone matrix." *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, California, 1991, pp. 394-403. (Accepted for publication in *J. of Algorithms.*)

[6] R. P. Brent. "The parallel evaluation of general arithmetic expressions." *J. of the ACM* 21 (2) (1974), pp. 201–206.

[7] R. Cole. "Parallel merge sort." *SIAM J. Comput.* 17 (4) (1988), pp. 770–785.

[8] P. J. de Rezende, D. T. Lee, and Y. F. Wu. "Rectilinear shortest paths in the presence of rectangles barriers." *Discrete Comput. Geom.* 4 (1989), pp. 41–53.

[9] H. ElGindy and P. Mitra. "An efficient parallel for computing the orthogonal shortest path among rectangular obstacles." Manuscript, January 1991.

[10] H. ElGindy and P. Mitra. Private communication.

[11] S. Guha and Q. Stout. Private communication.

[12] T. M. Nicholl, D. T. Lee, Y. Z. Liao, and C. K. Wong. "On the X-Y convex hull of a set of X-Y polygons." *BIT* 23 (4) (1983), pp. 456–471.

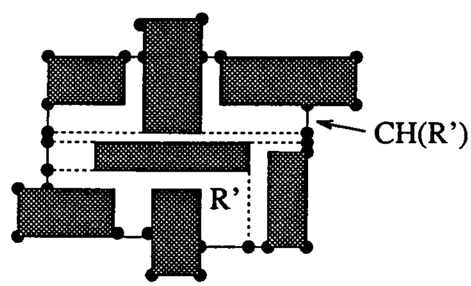

Figure 1

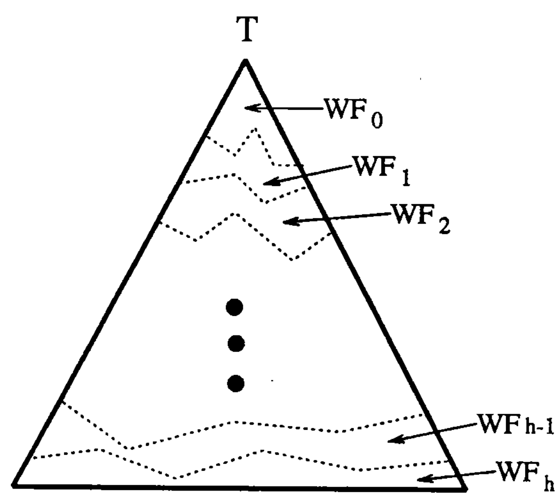

Figure 2