Carleton University, School of Computer Science

95.508 Computational Geometry, Winter Term 1997

Project:

In Introduction to Parallel Geometric Shortest Path Algorithms in Two Dimensions

Silvia Götz

April 25, 1997

Abstract

Geometric shortest path problems in two dimensions aks for a shortest path between points in a plane. The structure of a geometric shortest path problem can vary widely. The shortest path between two vertices or all shortest path between a source point and all vertices are computed. Different metrices are considered and the region that contains the query points and the shortest path can have differend shapes. Moreover, given some obstacles one regards the shortest path that avoids the obstacles. All of the problems that are presented in this paper have real-world applications. To reduce the computation time, nowadays, often the computational power of a parallel computer is used. Geometric shortest path problems are fundamental problems in computational geometry. In geometric applications they appear frequently and on large data sets. Problems with obstacles (which occur most naturally) are more time intensive to compute. All of this makes the analysis of parallizing geometric shortest path problems highly interesting.

This paper gives an overview of the parallel results for geometric shortest path problems in the plane. The main parallel algorithms are descriped at a top level giving the basic ideas. Results of algorithms are updated when an improvement of an algorithm for a building blocks are available, e.g. parallel triangulation. We also state open problems to direct further research.

Contents

1 Introduction

The problem of determining the shortest path between points in the plane is a fundamental problem in computational geometry. Like other graph theoretic problems in a geometric setting, geometric shortest path problems and their solution differ greatly from their graph theoretical counterparts. The geometry of regions and point sets makes different approaches possible. However, many algorithms also utilize graphs and their properties, especially of trees, e.g. the *dual* of a triangulation. Geometric shortest path problems occur naturally in real-word application, e.g. in robotics and VSLI design. Geographic information systems (GIS) operate on large data sets and shortest path queries are one of the most frequently tasks. However, for a lot of shortest path problems in real life a plane, even with obstacles, does not model the problem accurately enough, so that shortest paths are considered on spatial data structures. But still, shortest path problems in the plane have their own field of application (e.g. broadcasting), serve as an approximation for more constraint domains and are part of a plethora of other algorithms (e.g., visibility problems).

There is a great variety of different types of geometric shortest path problems. The region in which the query points lie and that the shortest path is not allowed to leave can have different shapes. For instance, besides an arbitrary simple polygon a *rectilinear* simple polygon is also of interest. *Rectilinear* segments are parallel to one of the coordinate axes (which we assume wlog to be horizontal and vertical). The definition is extended naturally to a path, polygon, etc. referring to the segments these objects consist of. Shortest path problems become more difficult, if some obstacles are given and the shortest path has to avoid these obstacles. In fact, for the general case of Eucidean metric and arbitrary polygonal obstacles, no efficient parallel solution has been found so far. But special cases, like rectilinear rectangular obstacles, have been shown to be easier to handle and occur naturally, e.g. in VLSI design and architecture. The problem of avoiding rectilinear obstacles is combined with looking for a rectilinear shortest path between the query points, i.e. a shortest path in L_1 or *Manhatten* metric avoiding the obstacles. Another metric is also interesting to consider, the *link* metric. The length of a path in the link metric is the number of segments (or links) it consists of. This metric is motivated by applications where moving along a straight line is cheap, whereas a turn is expensive. This is the case, e.g., for motions of a robot when considering the time. The link metric is often considered in the rectilinear setting, so that the *rectilinear link distance* between two points is the minimal number of segments on a rectilinear path between the two points. We will use the term *visible* if two points can be connected by a segment in the according metric without crossing the border of outer region or an obstacle.

The problem of shortest path problems in the plane have been well studied sequentially, e.g.,[Lee78][LePr84][LoWe79][SSch86][AMS91][Berg91][Ke89][ReSt87][Suri86]. There are not that many parallel algorithms for geometric shortest paths, but for some shortest path problems optimal solutions could be found [LMSa95][Hers92]. The frequency and the size of the geometric path problems make the utilization of parallel computing very attractive. Problems with obstacles are even more time intensive and, thus, even more interesting to parallelize. The used model in the presented algorithms is the PRAM (parallel random access machine), where the processors have a unique identity and share a common memory. The processors communicate through the shared memory and several types of the PRAM can be destinguished according to the constraints of common read and write accesses of the same memory location. The algorithms that we will mention use either the EREW (exclusive read, exclusive write) or the CREW (concurrent read, exclusive write) PRAM model.

In parallel the straight-forward approach for shortest path problems in polygons is computing the visibility graph and " matrix multiplication" to compute the transitive closure. The visiblity connects two vertices if they are visible from each other by an edge which has the weight of the distance. This results in a matrix with the Euclidean distances between each pair of vertices. The standard matrix multiplication is adapted for shortets path computation, i.e., *min*-function instead of + and + instead of $\cdot$, so that $C[i,j] = \min_k A[i,k] + A[k,j]$ in $C = A * B$. By applying matrix multiplication it the above sense, the transitive closure can be computed. This leads to an parallel algorithm that requires $O(\log^2 n)$ time and matrix-multiplication number of processors (currently around $O(n^{2.5})$). But better solutions have been found for a lot of shortest path problems.

This paper gives an overview of the existing parallel algorithms for shortest path problems and outline their main ideas. Some of the algorithms are updated due to used building blocks which have been improved, like parallel triangulation and visibility map computations. A lot of shortest path problems with obstacles are unsolved in parallel. Therefore, this paper aims to state open problem to direct further research.

The remainder of the paper is organized as follows. In Section 2 algorithms are presented that deal with Shortest Paths in a Simple Polygon in the Euclidean, Manhatten and link-metric. Parallel shortest Path algorithms that avoid obstacles are discussed in Section 3. Finally, some conclusions are given.

2 Shortest Paths in a Simple Polygon without Obstacles

2.1 Shortest Path in an Arbitrary Simple Polygon between Two Points

The *Interior Shortest Path (ISP)* problem is to find the shortest path between two points, a source s and a destination t, inside an n-vertex simple polygon (see Figure 1). The used metric is called *geodesic*, i.e., paths are measured in Euclidean metric, but they must not cross the boundary of P (they may touch it). Sequentially, in $O(n)$ time a data structure can be constructed so that the shortest path length queries between two points in P can be answered in $O(\log n)$ time [GuHe89]. This optimal result became possible by Chazelle's linear time triangulation algorithm for simple polygons [Chaz91]. ElGindy solved the ISP probem for monotone polygons P on a $O(n)$ processor CREW PRAM in $O(\log^2 n)$ time [ElGi86]. This result was improved in both applicability and time by ElGindy and Goodrich in [ElGo88]. Their algorithm requires on the same model $O(\log n)$ time using $O(n)$ processors. It proceeds

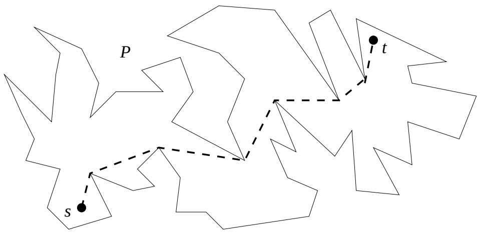

Figure 1: The Interior Shortest Path (ISP) problem: Find the shortest path between s and t interior a simple polygon P.

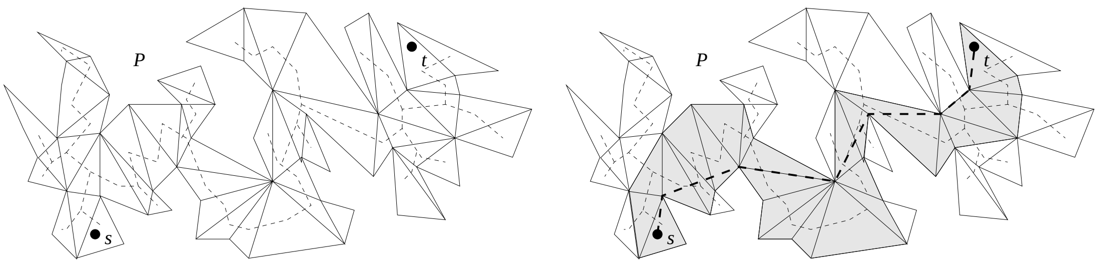

Figure 2: The triangulation T of polygon P, the dual D of T and the triangulated sleeve from s to t.

in two phases. Phase 1 starts quite commonly for these problems with a triangulation T of the polygon and constructing the dual D of the triangulation which is a tree (since P is a simple polygon). The *dual* of a decomposition is a graph where each node corresponds to a component (e.g. a triagle) and and two nodes are connected iff the the corresponding components share an edge. Let t_s (t_d) dennote the triangles which contain s (t) and v_s (v_d) the corresponding nodes in D. Then, the path in the graph D between v_s and v_d is determined, which is easy since D is a tree. This path in D corresponds to a triangulated polygon in T when considering the according triangles. This triangulated region is called a triangulated *sleeve* (see Figure 2). The sleeve is interesting to consider because the shortest path between s and p is completely contained in it, so that we get a narrow "area of interest". One can observe that the shortest path between two points in a simple polygon consists of a sequence of tangents to the boudary of P and segments to s and t (if s and t are not visible from each other). Therefore, one can approach the problem by compute the shortest paths between the vertices of the sleeve between s and t. The second phase of the algorithm makes use of that and is a divide-and-conquer approach which determines the shortest path between s and t by recursively computing the shortest path between the endpoints of two diagonals e_i, e_j of the sleeve. To be more specifically, only the shortest path between the two top vertices and the two bottom vertices have to be computed (we call them also top and bottom chains). The set of the two chains is called *hourglass* $H(e_i, e_j)$ (Figure 3).[1]Vertices of an hourglass are stored in clockwise order. The sleeve is devided into two pieces by taking the

[1]In fact, in [ElGo88] the four point-to-point shortes paths and the maximal common path of the four is stored, too, but this additional information is not necessary [GSG92]. The term hourglass was not used in [ElGo88].

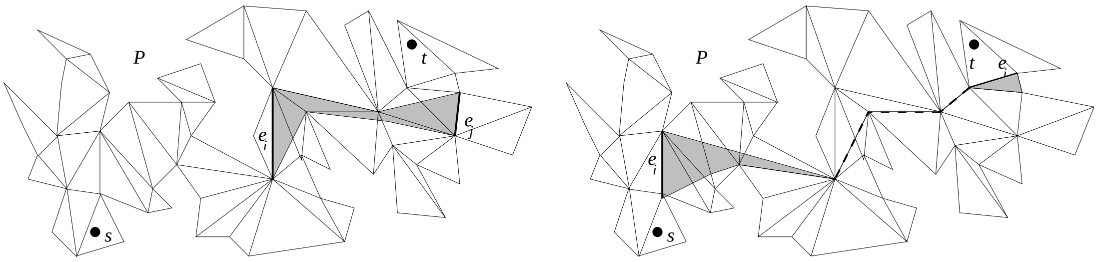

Figure 3: An hourglass consist of an upper and an lower chain, which not common parts are convex. Left: an hourglass which has no common part (*open)*, right: the upper and lower chain share edges (*closed* hourglass).

median diagonal. Two adjacent hourglasses are merged together using cross and supporting tangents and "cut and paste" of the appropriate parts of the old paths and the tangents. A *cross* tangent is a tangent between a top and a bottom chain, whereas a *supporting* tangent is a tangent between two top or two bottom chains. Algorithm 1 shows a top-level description of the algorithm.

Algorithm 1. *Computing the shortest path between two points interior an n-vertex simple polygon*

Input: A simple polygon P, two points s and t interior P.
Output: A list of vertices which form the shortest path from s to t in P.

Phase I:

1. Compute triangulation T of polygon P.
2. Construct the dual graph D of T.
3. Let v_s and v_d denote the nodes of D corresponding to the triangles where s and t lie in. Compute the simple path S' from v_x to v_d on D using [TaVi85];
3. Consider the to S' corresponding triangles (triangulated sleeve). Order diagonals $e_1, \ldots, e_m$ of the triangulated sleeve so that $e_i < e_j$ iff e_i is closer to s than e_j.

Phase II: Compute hourglass $H(e_1, e_m)$ recursively as follows:

1. If $e_1 = e_m$ then return endpoints of diagonal.
2. Find median diagonal e_j and compute $H(e_1, e_j)$ and $H(e_j, e_m)$ recursively.
3. Merge $H(e_1, e_j)$ and $H(e_j, e_m)$ to $H(e_1, e_m)$ by "cut and paste" of cross and supporting tangents to the chains and parts of $H(e_1, e_j)$ and $H(e_j, e_m)$.
4. Merge $H(e_1, e_m)$ with s and t;

end

Crucial for the algorithm is that parts of a chain of an hourglass which do not belong to the common path are convex. It makes the efficient computation of the tangents between upper and lower chains possible using [AtGo86]. Figure 4 shows the merging step when there is no common part of the chains. The algorithm needs constant time with $O(n)$ processors for each of the $O(\log n)$ recursion levels, so that Phase II requires $O(\log n)$ time using $O(n)$ processors. Phase I performs in the same bounds. The used triangulation

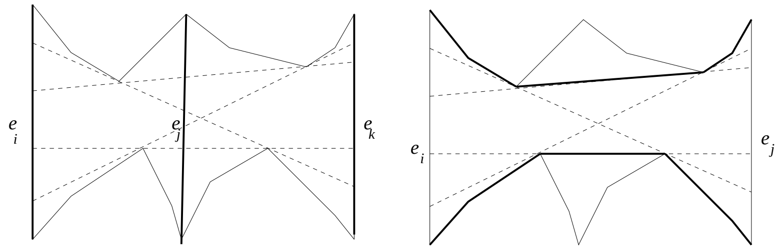

Figure 4: Merging hourglasses $H(e_i, e_k)$ and $H(e_k, e_j)$ when the common part of the upper and lower chain beteewn edges e_i and e_j is empty. The cross and supporting tangents are shown left and the resulting shortest paths between the two upper and lower vertices can be seen right. They are computed using "cut and paste" of the tangents and the old paths. The two new chains form the hourglass $H(e_i, e_j)$

algorithms of [Good89] was improved by several algorithms [CCT91] and optimaly solved [ACG87](det.)[ReSe92](rand.), but does not affect the complexity of the just presented algorithm. However, the ongoing progress in trianguation of a simple polygon encouraged optimal algorithms on already triangulated polygons. Such an algorithm has been developed by Goodrich *et al.*[GSG92]. It takes a triagulated polygon as input and needs $O(\log n)$ time on a *CREW PRAM* using $O(n/\log n)$ processors to construct a data-structure, called the *stratified decomposition tree*. The data-structure allows to construct a representation of the shortest path between two points inside in a simple polygon in $O(logn)$ time. Hence, with an optimal parallel triangulation algorithm the algorithm of [GSG92] is also optimal for non-triagulated simple polygons.

We describe now the data structure, the stratified decompositon tree which is based on the data structure of Guibas and Hershberger [GuHe89][Hers91]. Chazelle's polygon-cutting theorem [Chaz82] shows that there is a diagonal of a simple polygon P, which splits P into subpolygons of roughly the same size ($\in [1/3, 2/3]$). Recursively removing such a diagonal for the subpolygons results in a balanched, hierarchical decomposition of P. This decomposition can be represented as a binary tree in which each node corresponds to a subpolygon, and the children of a node are the subpolygons into which the polygon is split. Also for each node the splitting diagonal is stored. This decompostion tree is now modified to the new datastructure by contracting the nodes at a bottom stripe (Figure 5, left) and by assigning some nodes in the tree additional information. These nodes form "layers" in the decompostion tree which gives the datastructure its name (Figure 5, right). Each node in such a layer knows (among other information) the hourglasses between diagonals on the next layer's nodes in its subtree. This information and computing it does not exceed the bounds because of the contraction of the bottom part of the tree. The stratified decompostion tree is preprocessed for planar point location [TamV89][CoZa90] and lowest common acestor computation [SchV88] which can be done optimally. A node of a layer can for all points in the corresponding polygon determine which subpolygons of the next layer contains them. But more than that, for points in different subpolygons of the next layer it stores already the hourglass of the diagonals that the path between the points crosses in this next layer.

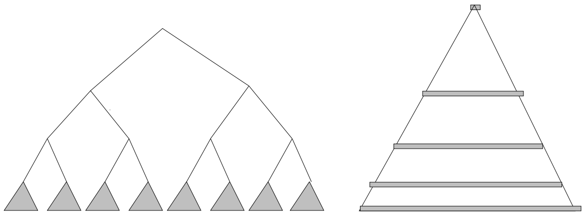

Figure 5: Left: A stripe on the bottom is contracted. Right: Some nodes get additional information. They are organized in layers.

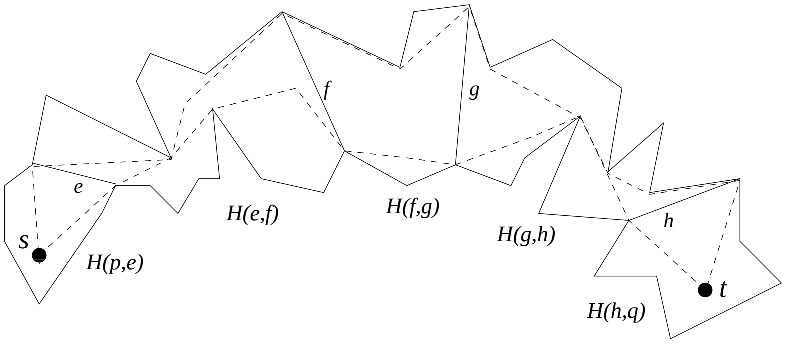

Figure 6: The contruction of $H(s,t)$: The hourglasses $H(s,e)$, $H(e,f)$, $H(f,g)$, $H(g,h)$, and $H(h,t)$ are needed.

With each layer another refinement in the corresponding polygons is made and additionally the shortest-path (hourglasses) between the subpolygons of this hierarchical level are already computed. This additional information enables a fast query response, but was on the other hand not too expensive to compute in advance since it is only done for layers and not for every node. The leaves of the tree are equipped with another datastructure which allows to apply a suboptimal algorithm. This does not affect the overall running time since the problems in the leafs are small.

The data structure of the stratified decomposition tree can be constructed in $O(\log n)$ time and $O(n)$ space with $O(n/\log n)$ processors in the CREW PRAM model. Using it shortest path queries between two points in the polygon P can be answered in $O(\log n)$ time by one processor. Processing a query is then more or less a merging of the already consisting hourglasses. Algorithm 2 describes the answering of a query (see Figure 6 for an illustration). The following table summarizes the algorithms for computing shortest path between two point inside a simple polygon using the Euclidean metric.

Algorithm 2. *Answering Shortest Path Queries*

Input: Two points s and t inside simple n-vertex polygon P, the stratified decomposition tree S of P.
Output: Representation of the shortest path between p and q.

1. Locate the subpolygons P_v and P_w on the leaf level of S in which s and t lie.

2. If $v = w$ then apply the suboptimal algorithm.
   Else
   2.1 Determine the paths to the least common ancestor z of v and w in S.
   2.2 Determine diagonals e and h that are on the boundary of P_v and P_w, respectively, and are intersected by the shortest path between s and t.
   2.3 Construct $H(s, e)$ and $H(h, d)$ with suboptimal algorithm.
   2.4 Locate diagonals f and g stored on P_z which are intersected by the shortest path from s to t.
   2.5 Construct $H(e, f)$ and $H(g, h)$ recusively by merging adjacent hourglasses using the technique of [ElGo88] described earlier.
   2.6 Merge $H(p, e)$, $H(e, f)$, $H(f, g)$, $H(g, h)$, and $H(h, q)$ binary [ElGo88].

end

Problem	Algorithm	Parallel time	No. of Proc.	Model
monotone polygon	[ElGi86]	$O(\log^2 n)$	$O(n)$	CREW
simple polygon	[ElGo88]	$\Theta(\log n)$	$O(n)$	CREW
simple polygon	[GSG92]	$\Theta(\log n)$	$\Theta(n/\log n)$	CREW

Table 1: Parallel algorithms for computing the Euclidean shortest path between two points interior a simple polygon. After building the data structure they all answer queries in $O(\log n)$ time using a single processor.

2.2 All Shortest Paths in an Arbitrary Simple Polygon

The *All Interior Shortest Paths (AISP)* problem consists of computing the shortest-paths between an interior source point s and each vertex of an n-vertex simple polygon P (Figure 7). More specifically, the shortest paths tree rooted at s is constructed which is the union of the shortest paths between s and all vertices of P. Since the shortest path from s to a vertice w consists besides s only of vertices of P it is sufficient to store for each vertex w of P the first edge incident to w of the shortest path to s. We denote this edge by $first(w)$ (Figure 7). The algorithm in [ElGo88] computes the shortest paths tree in $O(\log^2 n)$ time on a *CREW PRAM* with $O(n)$ processors. To get the length of the shortest paths requires additional $O(\log n)$ time. Instead of splitting the polygon by a diagonal one can also split it nicely (in 3 subpolygons) by removing a triangle. This can be done by computing the *centroid* of the dual D of a triangulation of P with the accelated method of Cole and Vishkin [CoVi88]. A *centroid* is a node of the tree D and when it is removed a the maximal size of the remaining subtrees is minimized. Recursively the shortest paths are computed to the vertices of the triangle in which s lies using the construction of hourglasses (like in the algorithm for *ISP*). Each of the $O(\log n)$ iterations needs $O(\log n)$ time.

This result was improved by using the stratified decompositon tree. Construction of the shortest path tree rooted at a node s to all vertices of P can be done in $O(\log n)$ time using $O(n)$ processors. Each vertex v of polygon P is assigned a processor. This processor computes the first edge $first(w)$ on the shortest path from w to s in a straight-forward

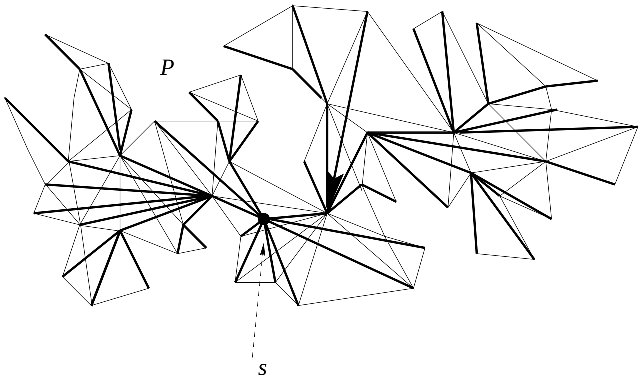

Figure 7: The All Interior Shortest Paths (AISP) problem: Construct the shortest path tree from s to all vertices of polygon P.

fashion using the stratified decomposition tree of P and the ISP algorithm. The $O(n)$ of processors are needed even when polygon P is already triangulated.

The $O(\log n)$ factor from optimal work can be avoided by using the algorithm of [Hers92] which reduces the number of required processors to $O(n/\log n)$. The algorithm is based on the stratified decomposition tree of [GSG92]. It uses one divide and conquer step, where the original problem of size n is devided into $O(n/\log n)$ subproblems of size $O(\log n)$. After that an opitmal algorithm [HeSn91] solves each of the $O(\log n)$-sized subproblems. The basic idea to get these $O(\log n)$ sized subproblems is simple. Along the boundary of P every $\log n$-th vertex p' is chosen. We compute in parallel the shortest path from source s to each p'. Considering two adjacent chosen vertices the region between their shortest paths to s and the boundary of P between these two nodes looks almost like a funnel. This region is called *pseudo-funnel* and is illustrated in Figure 8. The part of a pseudo-funnel which is the boundary of P between the two selected vertices is the *base* of the pseudo-funnel and the chain between selected vertex and the apex of the pseudo-funnel is the *side path* of the funnel. It can be shown that there is a pseudo-funnel where a vertex p is on the base and its parent in the shortest-path tree is on the side pathof this funnel. Unfortunately, the number of the vertices on the side path of these pseudo-funnel is only bounded by n which is too much to treat them independently in parallel. However, the total size of the funnels is $O(n)$. We want to have pseudo-funnels which sides are only of sides $O(\log n)$. Therefore, each $\log n$-th vertex of the sides path of each pseudo-funnel is now chosen and used to divide the pseudo-funnels into smaller ones. Using these small pseudo-funnels we define a new set of pseudo-funnels where each pseudo-funnel is of size $O(\log n)$. After a triagulation of the funnels the optimal algorithm of [HeSn91] computes for each vertex in a pseudo-funnel the parent in the shortest-path tree.

2.3 Shortest Paths in Link-Metric

In some areas, like motion planning or VSLI design, making a turn in a path from source s to destination t is more expensive than moving along a straight line. This is the case,

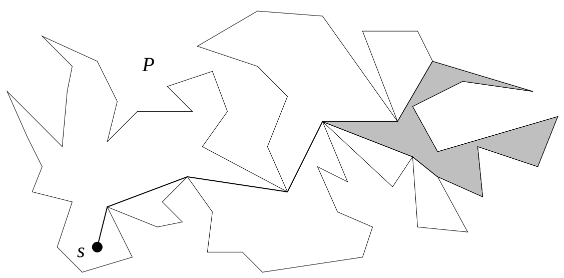

Figure 8: A pseudo-funnel.

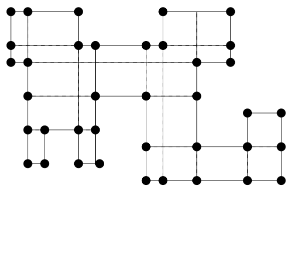

Figure 9: The visibility map of a rectilinear horizontal polygon.

e.g. for the time a robot needs to go from s to t when the straight lines are not "too long". These applications initiate the consideration of paths which cost is measured by the turns they make or, similarly, of how many segments (links) they consist.

A segment is called *rectilinear*, if it is parallel to one of the coordinate axes (which are wlog assumed to be horizontal and vertical). This term is extended to rectangles, polygons, paths, etc. in a straight-forward fashion, i.e., they are called rectilinear if the segments, that they consist of, are rectilinear. Considering the link-metric rectilinear paths often occur in practise, e.g. VLSI design. Rectilinear simple polygons are common approximations for simple polygons, and do also occur naturally, e.g. in architecture. Then, the rectilinear *link-distance* between two points s and t (for short $LD(s, d)$) is defined as the minimum number of links of a rectilinear path between s and t. In this section we use the term shortest path for rectilinear link-distance shortest path.

Link-distance problems seem to be more difficult than shortest path problems using the geodesic metric since the shortest path in the link metric is not unique. In [LMSa95] optimal parallel solutions for several link-distance problems in a simple rectilinear polygon P are presented when P is already trapezoided. A rectilinear simple polygon is *trapezoided* if the horizontal and vertical visibility maps are given. That is, each edge of polygon P is extended to the interior of the polygon until it hits the boundary of P (see Figure 9). This subdevision of P can be seen analog to the triangulation in the case of arbitrary polygons and non-rectangular paths. In [LMSa95] these extension points are inserted as vertices of the polygon. Then, given the trapezoidation of the n-vertex polygon P, among others, the following results are obtained. The link-distances and the length of L_1-shortest paths from a source s in P to all vertices of P can be computed in $O(\log n)$ time and $O(n/\log n)$ processors on an EREW PRAM. And also the construction of a data structure that supports rectilinear link-distance queries can be computed within the same bounds. Using this data structure and a single processor a query for the shortest path and can be answered in $O(\log n)$ time. The algorithms are work optimal since the sequential lower bound is linear (resp., $\log n$ for anwering queries). The visibility maps, required as input, can be computed for a arbitrary simple triagulated polygon in optimal $O(\log n)$ time with $O(n/\log n)$ processos on the CREW PRAM. With an optimal triangulation algorithm [ACG87][ReSe92] the visibility maps can be computed optimally. This answeres the (in [LMSa95]) open question of the optimality of these algorithms for a non trapezoided simple rectangular polygons.

We scetch now the algorithm to compute the link-distance shortest paths from a point s in P to all vertices of P. This algorithm improves upon the result of [GhMa92] of $O(\log^2 n \log\log n)$ time using $O(n)$ processors on a CREW PRAM. The algorithm in [LMSa95] makes use of the distance between diagonals in P. Roughly, a *diagonal* combines horizontally or vertically two point on the boundary of P of wich at least one is vertex. Clearly, some of the diagonals overlap each other. A *maximal diagonal* is a diagonal which is not properly contained in another diagonal. Analog, a *minimal diagonal* is a diagonal that does not properly contain another diagonal. The link-distance from a diagonal d to a vertex v is defined as the miminum link-distance of any point of d to v. And the link-distance between two segments is the minimum link-distance between any pair of points between the two segments. The following gives a top-level outline of the algorithm.

Algorithm 3. *Computing the link-distances from a point s in P to all vertices of P*

Input: A trapezoided rectilinear simple polygon, and a source s in P.
Output: Link-distance from s to all vertices of P.

1. Compute the horizontal and vertical segments through point s to the boundary of P (so-called *chords*).
2. Compute for each of the two chords containing p the link-distances to every vertex of P (which is essentially the same as the distances form a minimal diagonal to all vertices and can be computed optimally). The minimum of the two values gives us the link-distances to p or a value which is less by one.
3. Compute the part (so-called *interval*) of each chord which can be reached by an optimal link-distance path. If s is not in the interval, one additional link is necessary.

end

Step 2 of the algorithm is quite envolved and only the intuition is given here. Let d be a horizonal minimal diagonal. (Proceed analog for the vertical case.) For each vertex v of P the following is done in parallel. Let e be the maximal horizontal diagonal including vertex v. And let e' be the most remote (from e) maximal diagonal where the first turning of a minimum link path from v to d could occur when starting vertically at v. Using a tree structure on the maximal horizontal diagonals which is rooted "towards" d and parallel lowest common ancestor computation the maximal horizontal diagonal e' can be determined. The link-distance $LD(d, v)$ between d and v can then be composed of the minimum of $LD(e, v)$

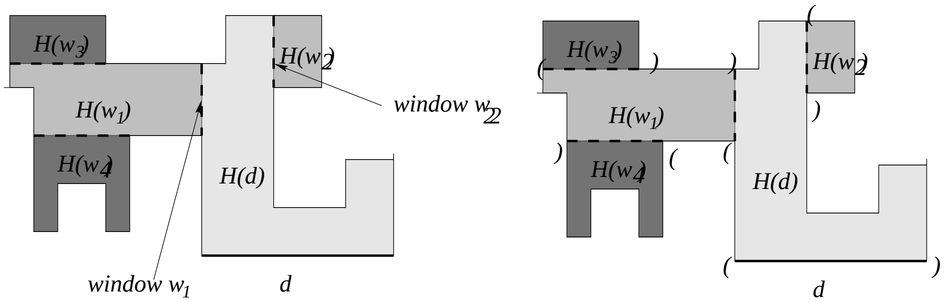

Figure 10: Left: The histogram partition of a simple rectilinear poligon in respect to diagonal d. The histograms that have the same link-distance to d have the same colour. The windows of a histogram form the base for other histograms that have an additional link-distance to d of one. Right: Assign paranthesis to the vertices and the inserted visibility map points according to the link-distance of the neigbour vertice/visibility map point. This leads to an optimal construction algorithm of the histograms.

and $LD(e', v)$ with minor changes. This is done separately for the subpolygons above and below d.

Next, we describe the data structure which supports optimally rectilinear link-distance queries. A partitioning of the simple rectilinear polgygon P into regions wich have the same link-distance from a fixed diagonal d leads to this data structure. The regions are called *histograms*. The partitioning can be visualized as follows. The histogram $H(d)$ with link-distance one from d is the region that is reachable form d with one link. d is called the *base* of histogram $H(d)$ which gives it its notation $H(d)$. The segments on the boundary of $H(d)$ which are not part of the boundary of the polygon P are called *windows*. These windows form the base for the subpolygons which are not contained in $H(d)$, i.e., the region which is reachable by one link from window w forms the histogram $H(w)$. It is a region that has link distance two from diagonal d. Figure 10 (left) gives an example of a histogram partition of a simple rectilinear polygon. The histogram partition of the trapezoided simple rectilinear polygon P can be optimally computed in $O(\log n)$ time and $O(n/\log n)$ processors. (The diagonal d splits P into two subpolygons and the algorithm proccesses the two subpolygons analog, so that we do not distinguish between them and call them P.) First, the link-distance from$d$ to all vertices of P (recall, that they include the inserted ones from the visibility maps) is computed (using algorithm 3). Paranthesis are assigned to the vertices according to their link-distance and according to the link-distance of their adjacent vertices (resp., visibility points) in P. After computing the matching parantheses [BSV88][LePe92] a circular list of vertices that belong to each histogram can be built which finishes the computing of the histogram partitioning. See Figure 10 (right) for an illustration.

It is crucial for the data structure, that for two query points s and t wich lie on different sides of a horziontal diagonal e the shortest link path between s and t is composed of their shortest path to e or a path from s (resp. t) to e that has one more link. Therefore, the link-distance between s and t can be composed of the link-distances $LD(s, e)$ and $LD(d, e)$ with minor changes. These changes depend on the intervals of diagonal e which can be reached by paths from s (resp, t) of length $LD(s,e)$ and of length $LD(s,e)+1$ (resp, LD(d,e) and LD(d,e)+1). The intervals are called fast intervel $I_{fast}(v,e)$ and slow interval $I_{slow}(v,e)$. In fact, one can compute optimally the fast and slow interval for each vertex v to a horizontal diagonal e. For this, it is important to know, that for every vertex v of P (with $LD(v,e) > 2$) there is a vertex v_{next} so that a shortest path from v to e can go through v_{next} and still reach all the elements of the fast interval $I_{fast}(v,e)$. An analog vertex $v_{next}2$ can be found for paths of additional length one and the slow interval $I_{slow}(v,e)$. The data structure to support optimally the answering of link-distance queries can now be constructed as described in Algorithm 4.

Algorithm 4. *Constructing the data structure to support link-distance queries optimally*

1. Construct histogram partition of P in respect to e..
2. Construct the dual tree T_H of the histogram partition of P.
3. Process tree T_H and built a data-structure to be able to answer lowest common ancestor queries optimally [SchV88].
4. Process histogram partition and built a data-structure for efficient point location [TamV89].
5. Construct a data structure to determine v_{next} and v_{next2} for query points (details omitted).

end

Using this data structure link-distance queries of two arbitrary points s and t in the trapezoided rectilinear polygon P can be answered in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM. The algorithm for answering a query first locates the histograms in which s and t fall. If s and t are in the same histogram the link-distance between s and t is easy to determine using vertical and horizontal segments passing through the query points. Otherwise, the lowest common acestor histogram H in T_H of the histograms that contain s and t is computed. Then, the fast and slow intervals of w_s (resp., w_d) from s (resp., t) are computed, where w_s (w_d) is the window of histogram H, so that w_s (w_d) lies in the histogram with base w_s (w_d). The link-distance $LD(s,d)$ can be composed of the link-distances $LD(s,w_s)$ and $LD(d,w_d)$ with some minor changes according to the slow and fast intervals of w_s and w_d and in respect to if the windows w_s and w_d are visible from each other. A summary of the results for all pairs shortest path problems is given in the following table.

Problem / Metric	Algorithm	Parallel Time	no. of proc.	model
simple polygon / L_2	[ElGo88]	$O(\log^2 n)$	$O(n)$	CREW
simple polygon / L_2	[GSG92]	$\Theta(\log n)$	$O(n)$	CREW
simple polygon / L_2	[Hers92]	$\Theta(\log n)$	$\Theta(n/\log n)$	CREW
rectil. (trap.) polygon / L_1	[LMSa95]	$\Theta(\log n)$	$\Theta(n/\log n)$	EREW^2
rectil. (trap.) polygon / link	[LMSa95]	$\Theta(\log n)$	$\Theta(n/\log n)$	EREW^2

Table 2: Parallel all pairs shortest path algorihtms.

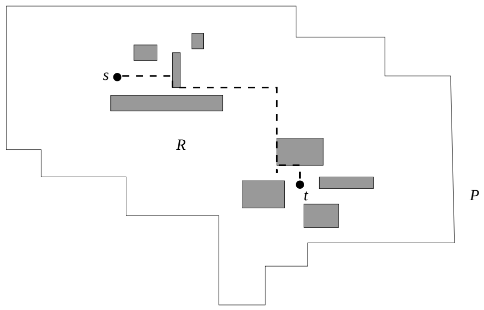

Figure 11: Finding rectilinear shortest paths from s to t in a simple convex rectilinear polygon P with rectilinear obstacles R.

3 Shortest Paths avoiding Obstacles

The problem here is to find a shortest path between two points s and t when a set of obstacles is given in the plane. A shortest path may touch but not cross an obstacle. For the case of Eucidian metric and polygonal obstacles in a simple polygon only the method of computing the visibility graph and the transitive closure of it is known. However, efficient algorithms can be given for special cases, i.e. for other metices and/or special shape and orientation of the obstacles.

3.1 Rectilinear Shortest Paths avoiding rectilinear rectangular obstacles

One common simplification of the problem is to consider not the distance in Euclidean metric but in the Manhatten or L_1 metric, where the path between two points consists only of rectilinear segments. Let for this section shortest path be the shortest path according to the L_1 metric.

In [AtCh91] the problem of rectilinear shortest paths between two points, source s and destination t, interior a simple rectilinear convex polygon P of size $O(n)$ is considered. (Convexity is measured in the sense of rectilinear paths.) In P lie n pairwise disjoint rectangular rectilinear obstacles. We use R to denote the set of obstancles (see Figure 11). The following results are shown.

1. Describtion of the shortest path can be computed in $O(log^2 n)$ time using
   - $O(n^2/\log^2 n)$ processors, if source and destination are on the boundary of polygon P,
   - $O(n^2/\log n)$ processors, if the source is a vertex of an obstacle and the destination is a vertex of the polygon P,
   - $O(n^2)$ processors, if both source and destination are obstacle vertices or arbitrary vertices.
2. These describtions enable a single processor to compute the length of a shortest path

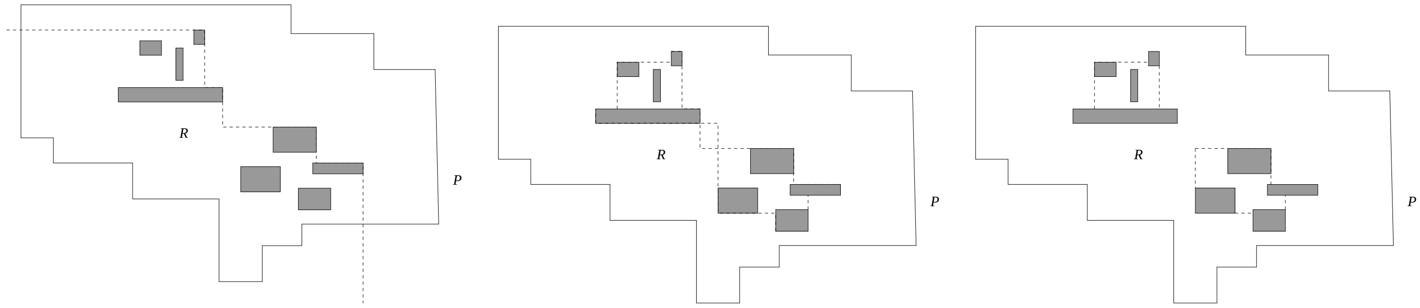

Figure 12: The union of closest staircases from the directions NW, NE,SW, and SE form the envelope $E(R)$ of the set of rectangles R. Left: he closest staircase from NE, middle: all closest staircases, right: the envelope of R

- between vertices in constant time, and
- between arbitrary points in $O(\log n)$ time.

3. The shortest path itself can be derived from its description in $O(\log n)$ time using $O(n/\log n)$ processors.
4. If the number of vertices of the polygon N is much bigger than the number of obstacles n, $O(\log N + \log^2 n)$ time and $O(N + n^2 \cdot f(n))$ processors are needed, where $f(n)$ is
   - 1 for any pair of points on the boundary of P,
   - $\log n$ for any pair of obstacle vertices and polygon vertices,

The algorithm uses a divide and conquer approach. For the divide step a so-called *staircase separator* S is constructed. That is, a rectilinear path in a shape of a staircase which partitions the rectangles in R into two subsets of maximal size $\frac{7}{8}n$, and the staircase S consists of $O(n)$ segments. The construction of S can be done in $O(\log n)$ time using $O(n)$ processors. It results in a decomposition of the obstacles in sets of roughly the same sizes. This yields in an recursion (and the corresponding tree recursion tree) of depth $O(\log n)$.

Let the *envelope* $E(R)$ of rectangles R be the possibly disconnected region of the plane which results of the "closest" staircases on R from NE (north-east), NW, SE, and SW (Figure 12,right). For e.g. closest staircase form NE means the lowest-leftmost one (see Figure 12, left). When $E(R)$ is not disconnected it equals the convex hull of the vertices of R. As in the section on the link-distance problems we consider instead of the vertices of Q the union of Q with the visibility map points and denote it by $B(Q)$. Region Q is either a polygon or an envelope of obstacles. The points in $B(Q)$ mark turn points when leaving or entering Q. The idea is as in the link-distance problems to compute the shortest path between the elements of $B(Q)$ instead between the vertices only. That leads to a recursive approach which efficently merges the shortest paths for B-points of the envelopes of two subsets of obstacles. We also can efficiently derive from the shortest path information for $B(Q)$ the shortest path between other boundary nodes of region Q. Let D_Q be the matrix that contains the length of the shortest path between all pairs of $B(Q)$. Algorithm 5 describes the top-level idea to compute the lenth of the shortest paths between vertices of P.

Algorithm 5. *Computing the shortest path length between vertices.*

Input: Set R of rectilinear rectangles included in polygon P, polygon P.
Output: Matrix D_Q, where $Q = E(R)$.

1. Sort vertices of rectangles.
2. Construct $Q = E(R)$ and $B(Q)$.
3. Compute recursively matrix D_Q:
   3.0 If R contains only one rectangle r then return D_r.
   3.1 Find staircase separator S and partition the set of rectangles R into R_1 an R_2; Let $Q_1 = E(R_1)$ and $Q_2 = E(R_2)$.
   3.2 Compute D_{Q_1} and D_{Q_2} recursively.
   3.3 Merge D_{Q_1} and D_{Q_2} to D_Q.
   3.4 Compute D_P from D_Q.

end

The data structure correspondig to the recursion tree T of algorithm 5 stores at each node additional information, like staircase separator, the two subsets of rectangles that results after the partition, etc. Therefore, a shortest path query between two points on the boundary of polygon P can be derived in logarithmic time using one processor and this data structure. For shortest path queries between a polygon vertice and an obstacle vertice, between two obstacle vertices or between arbitrary points in P more information for each node in T has to be computed and stored, which results in more work and, therefore, more processors (because the time stays the same). For instance, more shortest path matrices have to be computed and stored.

The merging of the shortest path according to the matrices $D_{E(R_1)}$ and $D_{E(R_2)}$ between the two subsets of Rectangles R_1 and R_2 is computed by the adapted matrix multiplication. Let M_{XY} the matrix that contains the length for shortest paths between each pair of vertices of set X and Y. When we multiply matrices M_{XY} and M_{YZ} together (in the above sense), the resulting matrix contains the length of shortest pathts between elements of X and Z when constraint to go through a vertice of Y. But using a vertex of a set $B(Q)$ as an intermediate vertice on the shortest path from a remote vertice to one interior $E(Q)$ does not affect the length of the shortest path. A crucial fact that is utilized by the algorithm is that matrices M_{XY},M_{YZ}, which have a certain monotonicity property (called *Monge* [AALF88] [AgPa88]) can be multiplied in $O(\log |Y|)$ time and $O(|X||X|)$ work. In the more complicated cases (i.e., shortest path not only between polygon vertices) chains are devided (which results in two matrices) to get two matrices which are Monge and exploit the cheap matrix multiplication. This scheme makes the efficient multiplication of shortest path matrices and, therefore, an efficient algorithm possible. For the cases of computing an shortest path from an arbitrary node p in P a preprocessing phase for efficient point location is necessary, since we need the obstacle that is right above p.

The idea for the case $|P| >> |R|$ is to devide the boundary of P into 8 chunks as shown in Figure 13. Then, $O(n)$ points (denoted by set C) for each chunk are determined by projection so that every (non trivial) shortest path goes through one of these points. Hence,

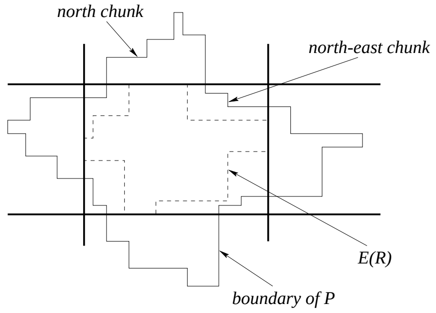

Figure 13: When the polygon P is much much bigger than the number of obstacles an implicit describtion of the shortest paths according to the chunks results in an efficient algorithm. The chunks of the boundary of P are built by horizontal and vertical lines along the extrema of the envelope of R.

the shortest-path matrix for C is a good describtion for the corresponding chunk. Through this implicit describtion a quadratic term in N in the work complexity can be avoided.

4 Conclusions

We have given an overview of parallel geometric shortest path algorithms. The basic ideas of the main results were described to give the reader an introduction to this area. The paper covers different metices as well as problems with given obstacles. For the following problems optimal parallel results could be achieved (some of them because of an optimal parallel triangulation or visibility map algorithm).

1. shortest path between two points interior a simple polygon,
2. all pairs shortest path tree between a source and all verices interior a simple polygon,
3. all pairs shortest L_1 path tree between a source and all vertices interior a rectilinear simple polygon,
4. all pairs shortest link-distance path tree between a source and all vertices interior a rectilinear simple polygon.

The results show that there is not much left to do in for settings without obstacles. But the field of shortest path problems where obstacles have to be avoided in the path is not well developed in parallel. Only for the very restrictive case of a rectilinear shortest path around rectilinear rectangular obstacles an efficient algorithm has been presented so far. Some sequential work is done on this field, e.g., [LePr84][CKV87]. One can think of applications to almost every kind of metric and obstacle shapes. Link-distance problems can also be considered in the case for non-rectilinear paths (e.g., the robot can not only make turns of 90 degrees). Of course, the general case for polygonal obstacles and Euclidean metric is of highly practical interest. Moreover, there are a wide range of related problems, most of them in visibility which we did not address in this paper. Dynamic schemes or parallel solutions for shortest path in $3d$ are also very attractive fields where not much work has been done yet. To solve shortest path problems with more restrictions or even for the difficult cases with obstacles approximation algorithms are of practical interest. To analyze the existing algorithms imperical is also valuable since shortest path problems are real-word problems. On the way to more practical parallel algorithms the existing algorithms can be examined if they can also be solved on other parallel computation models, e.g., the EREW PRAM model or on more practical ones like the coarse-grained models (BSP, LogP, CGM, ...).

References

[AgPa88] A. Aggrarwal, J. Park, "Notes on Searching in Multidimensional Monotone arrays (preliminary version)", *Proc. 29th IEEE Symposium FOCS*, pp. 497-512, 1988.

[AALF88] A. Apostolico, M. J. Atallah, L. L. Larnmore, J. S. McFaddin, "Efficient Parallel Algorithms for String Editing and Related Problems", *Proc. 26th Allerton Conf.*, pp. 253-263, 1988.

[AMS91] E. M. Arkin, J. S. B. Mitchell, S. Suri, "Link Queries and Polygon Approximation", *Proc. 3rd ACM SIAM Symposium on Discrete Algorithms*, pp. 269-279, 1991.

[AtGo86] M. J. Atallah, M. T. Goodrich, "Parallel Algorithms for Some Functions of Two Convex Polygons", *24th Annual Allerton Conf. Commun. Control Comput.*, Urbana-Champaign, pp 58-767, 1986.

[ACG87] M. J. Atallah, R. Cole, M. T. Goodrich, "Cascading divide-and-conquer: a technique for designing parallel algorithms", *Proc. of the 28th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 151-160, 1987.

[AtCh91] M. J. Atallah, D. Z. Chen, "Parallel Rectilinear Shortest Pahts with Rectangular Obstacles", Comput. Geom. Theory Appl., Vol.1, pp.79-113, 1991.

[Berg91] M. de Berg, "On Rectilinear Link Distance", *Computational Geometry: Theory and Applications*, Vol. 1, pp. 13-34, 1991.

[BSV88] O. Berkman, B. Schieber, U. Vishkin, "Some Doubly Logarithmic Optimal Parallel Algorithms Based on Finding all Nearest Smaller Values", Technical Report UMIACS-TR-88-79, University of Maryland, 1988.

[Chaz82] B. Chazelle, "A Theorem on Polygon Cutting with Applications", *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pp. 339-349, 1982.

[Chaz91] B. Chazelle, "Triangulating a Simple Polygon in Linear Time", *Discrete and Computational Geometry*, Vol. 4, pp. 485-524, 1991.

[CKV87] K. L. Clarkson, S. Kapoor, P. M. Vaidya, "Rectilinear Shortest Paths through Polygonal Obstacles in $O(n \log^2 n)$ time", *3rd Annual Symposium on Computational Geometry*, pp. 251-257, 1987.

[CCT91] K. Clarkson, R. Cole, R. Tarjan, "Randomized Parallel Algorithms for Trapezoidal Diagrams", *Proc. of the 7th ACM Symposium on Computational Geometry*, 1991.

[CoZa90] R. Cole, O. Jajicek, "An Optimal Parallel Algorithm for Building a Data Structure for Planar Point Location", *Parallel Distrib. Comput.*, Vol. 8, pp. 280-285, 1990.

[CoVi88] R. Cole, U. Vishkin, "The Accelerated Centroid Decomposition Technique for Optimal Parallel Tree Evaluation in Logarithmic Time", *Algorithmica*, Vol. 3, pp. 329-346, 1988.

[ElGi86] H. ElGindy, "A Parallel Algorithm for the Shortest Path Problem in Monotone Polygons", Technical Report MS-DIS-86-49, Deptartment of Computer and Information Science, Faculty of Engineering and Applied Science, University of Pennsylvania, Philadelphia, May 1986.

[ElGo88] H. ElGindy, M. T. Goodrich, "Parallel Algorithms for Shortest Path Problems in Polygons", *The Visual Computer*, Vol. 3, pp. 371-378, 1988.

[Good89] M. T. Goodrich, "Triangulating a Polygon in Parallel", *Journal of Algorithms*, Vol. 10, pp. 327-351, 1989.

[Good92] M. T. Goodrich, "Planar Separators and Parallel Polygon Triangulation", *Proc. ACM Symposium on Theory of Computing*, pp. 507-515, 1992.

[GSG92] M. T. Goodrich, S. B. Shauck, S. Guha, "Parallel Methods for Visibility and Shortest Path Problems in Simple Polygons", *Algorithmica*, Vol. 8, pp. 461-486, 1992.

[GuHe89] L. J. Guibas, J. Hershberger, "Optimal Shortest Path Queries in a Simple Polygon", *J. Comput. System Science*, Vol. 39, pp. 126-152, 1989.

[GhMa92] S. K. Ghosh, A. Maheshwari, "Parallel Algorithms for all minimum link paths and link center problems, *SWAT 92*, Lecture Notes in Computer Science, Vol. 621, pp. 106-117, 1992.

[Hers91] J. Hershberger, "A new Data Structure for Shortest Path queries in a simple polygon", *Inf. Process. Letters*, Vol. 38, pp. 231-235, 1991.

[HeSn91] J. Hersherber, J. Snoeyink, "Computing Minimum Length Path of a Given Homotopy Class", *Proc. of the 2nd Workshop on Algorithms and Data Sturctures*, pp. 331-342, 1991.

[Hers92] J. Hershberger, "Optimal Parallel Aglrotihms for Triangulated Simple Polygons", *Proc. 8th ACM Symposium on Computational Geometry*, pp. 33-42.

[Ke89] Y. Ke, "An Efficient Algorithm for link-distance Problems", *Proc. 5th ACM Symposium on Computational Geometry*, pp. 69-78, 1989.

[Lee78] D. T. Lee, "Proximity and reachability in the plane", Ph. D. Thesis, Tech. Report ACT-12, Univ. of Illinois, 1978.

[LePr84] D. T. Lee, F. P. Preparata, "Euclidean Shortest Paths in the Presence of Rectilinear Barriers, *Networks*, Vol. 14, pp. 393-410, 1984.

[LePe92] C. Levcopoulos, O. Petersson, "Matching Paraenthesis in Parallel", *Discrete and Applied mathematics*, Vol. 40, pp. 423-431, 1992.

[LMSa95] A. Lingas, A. Maheshwari, J.-R. Sack, "Optimal Parallel algorithms for Rectilinear Link-Distance Problems", *Algorithmica*, vol. 14, pp. 261-289, 1995.

[LoWe79] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyghedral obstacles", *Proc. ACM*, pp. 560-570, 1979.

[ReSt87] J. H. Reif, J. A. Storer, "Minimizing Turns for Discrete Movement in the Interior of a Polygon", *IEEE Journal of Robotics and Automation*, Vol 3, pp. 182-19 3, 1987.

[ReSe92] J.J. Reif, S. Sen, "Optimal Randomized Parallel Algorithms for Computational Geometry", *Algorithmica*, Vol. 7, pp; 91-117, 1992.

[SchV88] B. Schieber ,U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization", *SIAM Journal on Computing*, Vol. 17, pp. 1253-1262, 1988.

[SSch86] M. Sharir, A. Schorr, "On Shortest Paths in Polyhedral Spaces", *SIAM Journal of Computing*, pp. 193-215, 1986.

[Suri86] S. Suri, "A Linear Time Algorithm for Minimum Link Path inside a Simple Polygon", *Computer Vision, Grahpics and Image Processign*, Vol. 35, pp. 99-110, 1986.

[TamV89] R. Tamassio, J. S. Vitter, "Optimal Parallel Algorithms for Transitive Closure and Point Location in Planar Structures", *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pp. 399-408, 1989.

[TaVi85] R. E. Tarjan, U. Vishkin, "An Efficient Parallel Biconnectivity Algorithm", *SIAM Journal on Computing*, Vol. 14, pp. 862-874, 1985.