Planar Spanners and Approximate Shortest Path Queries among Obstacles in the Plane

Srinivasa Arikati[1*], Danny Z. Chen[2**], L. Paul Chew[3***], Gautam Das[1*], Michiel Smid[4†], and Christos D. Zaroliagis[5‡]

[1] Math Sciences Dept, The University of Memphis, Memphis, TN 38152, USA
[2] Dept of Computer Sc. and Eng, Univ. of Notre Dame, Notre Dame, IN 46556, USA
[3] Dept of Computer Sc, Upson Hall, Cornell University, Ithaca, NY 14853, USA
[4] Dept of Computer Sc, King's College London, Strand, London WC2R 2LS, UK
[5] Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

Abstract. We consider the problem of finding an obstacle-avoiding path between two points s and t in the plane, amidst a set of disjoint polygonal obstacles with a total of n vertices. The length of this path should be within a small constant factor c of the length of the shortest possible obstacle-avoiding s-t path measured in the L_p-metric. Such an approximate shortest path is called a *c-short path*, or a short path with *stretch factor c*. The goal is to preprocess the obstacle-scattered plane by creating an efficient data structure that enables fast reporting of a c-short path (or its length). In this paper, we give a family of algorithms for the above problem that achieve an interesting trade-off between the stretch factor, the query time and the preprocessing bounds. Our main results are algorithms that achieve logarithmic length query time, after subquadratic time and space preprocessing.

1 Introduction

Given a set of disjoint polygonal obstacles with a total of n vertices in the plane, the (geometric) *shortest paths problem* is that of finding a path between two points s and t (henceforth shortest s-t path) in the plane that does not intersect the interior of any obstacle, and that has the minimum length measured in the

[*] E-mail: `{arikatis, dasg}@next1.msci.memphis.edu`. Part of this work was done while the authors were with the Max-Planck-Institut für Informatik in Saarbrücken, Germany. Gautam Das was partially supported by NSF Grant CCR-9306822.

[**] The research of this author was supported in part by the National Science Foundation under Grant CCR-9623585.

[***] E-mail: `chew@cs.cornell.edu`. This author was supported by ONR Grant N00014-89-J-1946, by ARPA under ONR contract N00014-88-K-0591, by the U.S. Army Research Office through the Math. Sciences Institute of Cornell Univ. under contract DAAL03-91-C-0027, and by the Cornell Theory Center which receives funding from its Corporate Research Institute, NSF, New York State, ARPA, NIH, and IBM.

[†] E-mail: `michiel@dcs.kcl.ac.uk`. Part of this work was done while the author was with the Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[‡] E-mail: `zaro@mpi-sb.mpg.de`. This author was partially supported by the EU ESPRIT LTR Project No. 20244 (ALCOM-IT).

L_p-metric for some integer p, $1 \leq p \leq \infty$. To be more precise, such a shortest path consists of straight-line segments, called *edges*, and the *length* of each edge is equal to the L_p-distance between its end points. The *length* of the entire path is defined as the sum of the lengths of its edges. Computing shortest paths is a fundamental topic in computational geometry because shortest paths problems appear in many application areas, such as robotics and VLSI design, and play vital roles in solving various geometric problems.

In this paper, we consider shortest paths problems in the plane, for a general L_p-metric. Natural special cases are the L_1 and L_2 (Euclidean) metrics. Both these metrics have been extensively studied, since they are important in practice and have a lot of applications. Note that when we refer to a path in the L_1-metric, we do not mean that this path uses only line segments that are parallel to a coordinate axis, but that the lengths of the segments of the paths are measured in the L_1 metric. There are several different versions of the shortest paths problem, depending on whether we ask for a shortest path between any two obstacle vertices s and t, or whether s and t can be arbitrary points in the obstacle-free space. In addition, we are often satisfied in many applications with an obstacle-avoiding path that is not necessarily shortest, but whose length is within a small constant factor c of the length of a shortest path. Such an approximate shortest path is called a *c-short path*, and the constant c is called the *stretch factor* of the path.

The first problem considered in this paper is that of answering *short path queries* in the L_p metric: Given a set of disjoint polygonal obstacles in the plane, construct an efficient data structure that enables a fast report of an s-t c-short path (or its length) between any pair of query points s and t (regardless whether they are arbitrary points or obstacle vertices).

Previous results for this problem are as follows. Clarkson [13] gave an algorithm for L_2 $(1+\epsilon)$-short path queries among polygonal obstacles, for any positive ϵ. His algorithm uses $O(n)$ space, and answers a short path query in $O(n \log n)$ time. As was indicated in [13] and shown in [5], it is possible to extend Clarkson's result as follows. In $O(n^2 \log n)$ time, an $O(n^2)$ space data structure can be constructed such that a length query can be answered in $O(\log n)$ time. Reporting an actual $(1+\epsilon)$-short path takes $O(\log n + L)$ time, where L is the number of edges of the path. Chen [5] presented an efficient data structure for L_2 $(6+\epsilon)$-short path queries among polygonal obstacles. His data structure requires $O(n \log n)$ space and $O(n^{3/2}/\sqrt{\log n})$ time to construct, and supports a time of $O(\log n)$ for a length query and an additional $O(L)$ time for reporting an actual path. Results on some special cases of short path queries can be found in [6]. In contrast, the (exact) shortest path queries problem typically takes more time and space to solve. There are several results on L_2 shortest path queries in a simple polygon [12, 20, 21, 22] and L_1 shortest path queries among various types of obstacles [3, 4, 7, 16]. Note that for even the seemingly simpler case of L_1 shortest path queries among multiple obstacles in the plane, all known data structures supporting a polylogarithmic length query time require $\Omega(n^2)$ space and time to construct.

In this paper, we improve all of the above results for the short path queries problem. We give a family of algorithms whose bounds are summarized in Table 1.

Short paths are closely related to the notion of a *spanner*, see e.g. [13, 5]. Given a set S of n points, a *τ-spanner* is a graph having the points of S as its vertices, such that for any two points s and t of S, there is an s-t path in the graph of length at most τ times the L_p-distance between s and t. The problem of constructing spanners has attracted a considerable amount of attention recently (see e.g. [1] and the references given there). In particular, the main goal is to construct spanners that contain a linear number of edges (and that possibly satisfy some other constraints [1]).

Stretch factor	Preprocessing Time	Space	Query Time
$c+\epsilon$	$O(n^2/\sqrt{r})$	$O(n^2/\sqrt{r})$	$O(\log n + \sqrt{r})$
$c+\epsilon$	$O(n \log n)$	$O(n)$	$O(n)$
$2c+\epsilon$	$O(n^{3/2})$	$O(n^{3/2})$	$O(\log n)$
$3c+\epsilon$	$O(n^{3/2}/\sqrt{\log n})$	$O(n \log n)$	$O(\log n)$

TABLE 1: Our results for the short path query problems. For the L_1-metric $c = 1$, for the Euclidean metric $c = \sqrt{2}$, and in general for the L_p-metric $c = 2^{(p-1)/p}$. The parameter ϵ is an arbitrarily small positive constant, whereas r is an arbitrary integer, such that $1 \leq r \leq n$. The actual short path can be output in an additional $O(L)$ time, where L is the number of edges of the reported path.

The second problem considered in this paper is the following *planar spanner* problem. Given a set of disjoint polygonal obstacles in the plane, construct a graph $G = (V, E)$ such that: (i) the set of obstacle vertices S is a subset of V; (ii) the edges of G are straight-line segments that do not intersect the interior of any obstacle; (iii) for any two obstacle vertices $s, t \in S$, there is an s-t path in G which is a τ-short path; and (iv) G is planar. If $V = S$, then we call G a *planar L_p τ-spanner*. Otherwise, if G contains additional vertices (called *Steiner vertices*), we call G a *planar Steiner L_p τ-spanner*. The real number $\tau \geq 1$, representing the stretch factor of short paths, is called the *stretch factor* of the spanner. There are several algorithms that construct planar L_2 τ-spanners in $O(n \log n)$ time [9, 10, 15]. The best known stretch factor is $\tau = 2$ [9, 10]. We are not aware of any previous spanners specifically constructed for the L_1 metric. Regarding the planar Steiner spanner problem, no previous results are known in any metric.

We present the following new results for the planar spanner problem.

- We prove (Section 2) that a planar L_1 2-spanner among polygonal obstacles can be constructed in $O(n \log n)$ time without using Steiner vertices. This result is not only optimal w.r.t. time, but also w.r.t. stretch factor: we show that there are sets of polygonal obstacles in the plane such that *any* planar L_1 τ-spanner which does not use Steiner vertices has a stretch factor $\tau \geq 2$.
- In view of the above result, if we want a planar L_1 spanner with a stretch factor less than 2, then we *have to* use Steiner vertices. In this case, we

prove (Section 3) that, for any $\epsilon > 0$, a planar Steiner L_1 $(1+\epsilon)$-spanner with $O(n)$ Steiner vertices can be constructed in $O(n \log n)$ time. In fact, the same construction produces a planar Steiner L_p $(2^{(p-1)/p} + \epsilon)$-spanner with $O(n)$ Steiner vertices. (The constants that appear in the Big-O bounds depend on ϵ.)

The first result on constructing a planar L_1 2-spanner is based on a *constrained Delaunay triangulation* [8] that uses a special convex distance function. This convex distance function is defined by a carefully chosen equilateral triangle in the L_1 metric whose shape is somewhat different from the standard equilateral triangles. We also construct examples to show that 2 is the lower bound of the stretch factor achieved by any planar L_1 spanner without using Steiner vertices.

The approach used for the second result is based on an algorithm of Arya *et al.* [2], that, given a set of points, constructs a subdivision of the plane into boxes. We first build this subdivision on the set of obstacle vertices. Then, each of the resulting $O(n)$ boxes is "gridded" in an appropriate way. Each box gives a constant number of Steiner vertices. Superimposing the obstacles onto this subdivision gives a planar Steiner L_1 $(1+\epsilon)$-spanner, which may have $\Omega(n^2)$ Steiner vertices. We show, however, that by carefully merging regions of the superimposed subdivision, we get a planar $(1+\epsilon)$-spanner with only a linear number of Steiner vertices. But constructing the spanner in this way would still take $\Omega(n^2)$ time. Fortunately, our algorithm avoids the costly merging procedure and manages to construct the spanner in only $O(n \log n)$ time.

Our results for short path queries given in Table 1 are based on the above results on planar spanners, and on the following two graph-theoretic results (Section 4) which are of independent interest:

I. Given an n-vertex (directed or undirected) planar graph G with nonnegative real edge weights, we can perform, for any $1 \le r \le n$, an $O(n^2/\sqrt{r})$ time and space preprocessing of G such that the length of a shortest path in G between any two vertices can be found in $O(\sqrt{r})$ time.
II. Given an n-vertex undirected planar graph G with nonnegative real edge weights, we can perform an $O(n^{3/2})$ time and space preprocessing of G such that the length of a 2-short path in G between any two vertices can be found in $O(\log n)$ time.

We can also output the actual shortest (or 2-short) path between the query vertices in an additional $O(L)$ time, where L is the number of edges of the reported path. Our algorithms for planar graphs improve (in one or another way) upon known previous results (see e.g. [5, 14, 18]).

2 Planar L_1 spanners without using Steiner vertices

We first prove that planar spanners (without Steiner vertices) for L_1 shortest paths can achieve a stretch factor no better than 2. Note that it is sufficient to show this fact on the simpler case with just point-obstacles in the plane.

Lemma 1. *There exist point sets in the plane such that any planar L_1 spanner that is a subgraph of the complete graph has a stretch factor ≥ 2.*

Proof. We first consider a simple case: A unit diamond in the plane with points a, b, c, and d as its vertices (i.e., the L_1 distance from each vertex to any of the other vertices of the unit diamond is 1). Let K_4 be a complete undirected graph for vertices a, b, c, and d (see Figure 1a). It is clear that K_4, in which we let each edge have a unit weight, is the complete graph modeling the exact L_1 shortest paths among the four vertices. Suppose we obtain a spanner of K_4 by removing an arbitrary edge (say, the edge (a, b)). Then the shortest path from a to b in this spanner becomes 2. Therefore, no proper subgraph of K_4 can approximate the shortest paths with a stretch factor better than 2. Consequently, the only spanner of K_4 with a stretch factor less than 2 is K_4 itself.

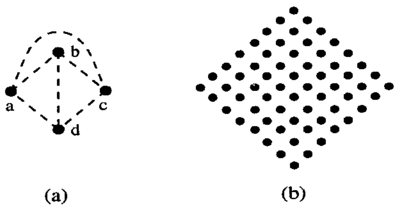

Fig. 1. (a) A K_4 on a unit diamond; (b) a diamond grid with n points.

Although the K_4 is a planar graph (Figure 1a), it plays a critical role in our following argument. Consider a set of n points forming a $\sqrt{n} \times \sqrt{n}$ "diamond grid" in the plane (see Figure 1b). A planar graph on these vertices has at most $3n - 6$ edges, while one can show that a larger number of edges $(4n - 6\sqrt{n} + 2)$ are required to make each unit diamond have stretch factor less than 2. □

The idea to efficiently construct a spanner with stretch factor exactly 2 is to use a type of the Constrained Delaunay Triangulation (CDT) using a distance based on a triangle [11, 8, 9]. The trick here is to find the right triangle-shape and the right triangle-orientation. Standard equilateral triangles that worked for the L_2 case, as described in [9, 10], fail to yield the desired planar L_1 2-spanner, so we turn to the following triangle shape: An isosceles triangle as depicted in Figure 2. Note that this triangle fits nicely within an L_1 circle (i.e., a unit diamond). In a sense, this is still an equilateral triangle: The L_1 length of the base is equal to 1 and each of the other two sides of the triangle also has its L_1 length equal to 1. We can prove the following.

Theorem 2. *Let S be the set of obstacle vertices for any set of disjoint, polygonal obstacles in the plane with $|S| = n$. There is a planar subgraph of the visibility graph, modeling the L_1 distances among the points in S, whose stretch factor is exactly* 2 *(i.e., an L_1* 2-spanner*). Furthermore, such a planar L_1 2-spanner can be constructed optimally in $O(n \log n)$ time.*

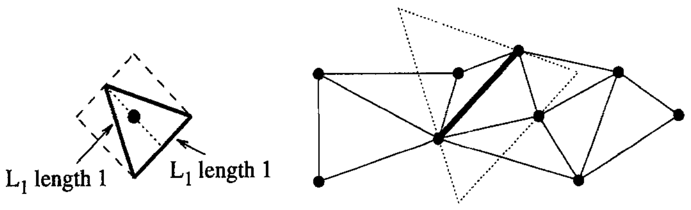

Fig. 2. Our distance-defining isosceles triangle and a CDT based on it. The dotted triangle represents an "empty circle."

3 Planar L_1 spanners with Steiner vertices

In this section we prove the following theorem.

Theorem 3. *Given a collection of disjoint polygonal obstacles on the plane with n vertices and any $\epsilon > 0$, a planar L_1 $(1+\epsilon)$-spanner with $O(n/\epsilon^2)$ Steiner vertices can be constructed in $O(n \log n + n/\epsilon^2)$ time.*

3.1 A Steiner spanner without obstacles

Let us start with the simpler problem of constructing a Steiner spanner for a set S of n points *without any obstacles.* (Note: in the rest of the section, the default metric is L_1). We frequently make use of a procedure called *interval.* Given a line segment xy and $r > 0$, *interval*(xy, r) starts from x and introduces Steiner vertices along xy such that the segment is broken into intervals of length r, except possibly for the last interval.

Arya *et al.* [2] describe a certain planar subdivision for solving nearest neighbor search problems, and we find it useful in constructing our Steiner spanner. This subdivision is a planar graph where each face is a connected region called a *cell.* The shapes of the cells have special significance, and are best described through the concept of *boxes.* A box is an axis-parallel rectangle such that the ratio of its longest side to its shortest side is at most 2. A cell is either a box (called a *box cell*), or the set-theoretic difference of two boxes, one contained within the other (called a *doughnut cell*). Furthermore, each doughnut cell is restricted as follows. For each side e' of the inner box, the orthogonal distance between e' and the corresponding side of the outer box is either zero, or greater than or equal to the length of e'. Finally, a box cell contains at most one point of S, whereas a doughnut cell contains no points of S. The following lemma can be derived from the results in [2].

Lemma 4. *Given a set S of n points, let B be any box containing them. Then, in $O(n \log n)$ time a subdivision D of B can be constructed such that, each cell of D is either a box cell or a doughnut cell, and the number of cells is $O(n)$.*

In constructing our spanner, we first construct a planar subdivision of any box B containing S, as in the above lemma. We then augment the graph of the subdivision by adding new Steiner vertices and edges within each cell as follows. Consider any box/doughnut cell. For every boundary edge xy, first perform $interval(xy, \epsilon l)$, where l is the length of shortest side of the box to which xy belongs. Then, from x, y and each of the Steiner vertices generated, shoot rays orthogonal to xy and directed into the interior of the cell, until they hit the cell's boundary, possibly creating new Steiner vertices. This process introduces $O(1/\epsilon)$ vertical and horizontal rays. If we further break the rays up by computing their intersections, we get a "grid" with $O(1/\epsilon^2)$ Steiner vertices and edges per cell. Note that the grid inside a doughnut cell is somewhat more complex than the one inside a box cell.

The resulting graph G is clearly planar and has $O(n/\epsilon^2)$ Steiner vertices. We now show that its stretch factor is $(1+\epsilon)$. Consider *any* path on the plane between points u and v of S, say $P(u,v)$. Let $P(u,v)$ be divided into the pieces $P_1, P_2, \ldots, P_k$, where each P_i is a maximal portion confined within a cell. Consider a particular P_i and let its end points be a and b. Note that a and b need not necessarily be vertices of G, although they lie on edges of G. We show that there is a path P_i' from a to b that "stays on" G, such that P_i' is at most $(1+\epsilon)$ times longer that P_i.

Consider one possible case. Suppose $2 \leq i \leq k-1$ and P_i is within a doughnut cell (with outer corners x, y, z and w as seen clockwise from top-left, corresponding inner corners x', y', z' and w', length of shortest side of outer box l, and length of shortest side of inner box l'). Suppose a is on xy and b is on $x'y'$. To construct P_i' we go from a along xy until we reach a Steiner vertex whose horizontal separation from b is within $\epsilon l'/2$ (such a Steiner vertex has to exist because of the upward ray shots from $x'y'$), then go down vertically until we reach $x'y'$, and finally go horizontally until we reach b. It is easily seen that the length of P_i' is at most $(1+\epsilon)$ times the L_1 distance between a and b.

A full case analysis (which we omit) proves the same for all possible types of P_i. If we merge the P_i's and eliminate any overlaps, we get a path $P'(u,v)$ composed of whole edges of G, which is at most $(1+\epsilon)$ times longer than $P(u,v)$.

The time taken to construct G is $O(n \log n + n/\epsilon^2)$, i.e. the time taken for the planar subdivision as well as the time taken to grid each cell.

3.2 A Steiner spanner amidst obstacles

Our solution for obstacles is similar, except that we use a more complicated subdivision. Let us treat each polygonal obstacle as a collection of *edge obstacles* (i.e. the boundary edges). Once the spanner has been computed, we can easily recognize and discard the portions that lie within the interiors of the original obstacles.

Let E be the given edge obstacles and let S be the set of their end points. Let B be a box containing them. The obstacles induce a natural subdivision of B into a single connected region where each obstacle edge is a "hole". Let this subdivision be D. We also compute another subdivision D' of B for the points of S as in Lemma 4. Let S' (E') be the vertices (edges) of D'. Define the subdivision D_1 as the superimposition of D on D'. The vertices of D_1 will be $S \cup S'$, and additional vertices created by intersecting edges of D and D'.

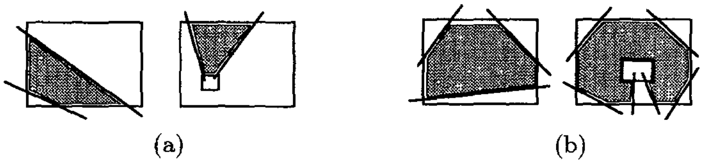

Fig. 3. (a) Red regions in D_1 (b) Blue regions in D_1

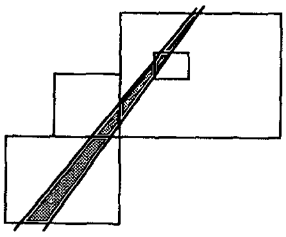

Fig. 4. A ladder in D_1

The regions of D_1 can be classified into two types. A *red region* is a quadrilateral such that none of the four vertices belong to $S \cup S'$ (see Figure 3(a)). Note that the number of red regions may be $\Omega(n^2)$. The rest of the regions are *blue regions*. Note that blue regions may or may not have boundary vertices from $S \cup S'$. As Figure 3(b) shows, the maximum number of boundary edges of a blue region can be 16. Define the *red graph* of D_1, as follows: Create a vertex for each red region; there is an edge between two vertices if the corresponding red regions share a boundary edge that is part of a box/doughnut cell. Each connected component of this graph is a simple chain, and all the red regions of the chain lie between the same two obstacle edges (Figure 4). The configuration resembles a *ladder*, with the obstacle edges forming the two sides, and parts of box/doughnut cell edges forming the *rungs*. Of course, some rungs may be horizontal while others may be vertical. Now *collapse* every ladder in D_1 (i.e., eliminate the intermediate rungs), yielding a single quadrilateral red region per ladder. Call the resulting subdivision D_2. (D_2 has the same blue regions as D_1.)

Lemma 5. *The subdivision D_2 has $O(n)$ vertices, edges and regions.*

Proof. We omit the details in this version. The idea is to show that there are $O(n)$ extremal rungs of ladders remaining in D_2. This is done by a charging technique, where each rung is charged to a vertex in $S \cup S'$ "just outside" its ladder. We can show that in this way, each vertex in $S \cup S'$ is charged at most a constant number of times. □

A crucial result of our paper (to be presented later) is an efficient and non-trivial algorithm for constructing D_2. For the moment, assume we have constructed D_2. We discard the red/blue regions that lie within obstacles, then grid the remaining regions as follows. For every blue/red region, for every boundary edge xy that is part of a box/doughnut cell, first perform $interval(xy, \epsilon l)$, where l is the length of the shortest side of the box to which xy belongs. Then, from x, y and each of the Steiner vertices generated, shoot rays orthogonal to xy and directed into the interior of the region, until they hit the region's boundary, possibly creating more Steiner vertices. Finally, compute the intersections of the horizontal and vertical rays, thus constructing a $O(1/\epsilon^2)$ grid for the region.

Clearly the eventual graph G is planar, and has $O(n/\epsilon^2)$ Steiner vertices. Although some edges may not be axis-parallel, the length of each is defined in the L_1 metric. The proof that its stretch factor is $(1+\epsilon)$ is structurally similar to the earlier proof without obstacles; essentially we show that in a red/blue region, between every a and b on boundary edges that are parts of box/doughnut cells, there is an $(1+\epsilon)$-short path that "stays on" G. The details are omitted.

Finally, we outline the efficient algorithm for constructing D_2. Clearly a naive algorithm which first constructs D_1 and then collapses ladders will take $\Omega(n^2)$ time. Our algorithm avoids constructing D_1. Let E'_h and E'_v be the set of horizontal and vertical edges of D'. Define D_{1h} (D_{1v}) as the superimposition of D with only E'_h (E'_v). As was done for D_1, define ladders for D_{1h} (D_{1v}) (note that in D_{1h} (D_{1v}) the rungs are horizontal (vertical)). Define D_{2h} (D_{2v}) as D_{1h} (D_{1v}), but with all ladders collapsed. Define D_3 as the superimposition of D_{2h} on D_{2v}. Define ladders for D_3. It can be shown that each ladder in D_3 has at most four rungs, and that D_2 is D_3 with all ladders collapsed.

The algorithm consists of four stages. In the first and second stages, we construct D_{2h} and D_{2v} respectively. In the third stage we construct D_3 (easy to do since the horizontal edges of D_{2h} do not intersect the vertical edges of D_{2v}). In the last stage we construct D_2 (easy to do by collapsing the ladders of D_3).

We give details of the first stage (the second stage is similar). It is implemented by an upward plane sweep which discovers the top rungs of all ladders in D_{2h}, followed by an identical downward sweep that discovers the bottom rungs.

Consider the upward sweep. At any instant, the sweep line intersects a subset of the obstacle edges, where a pair of adjacent edges in the left-to-right order define an interval on the sweep line. Consider any interval $I = (l, r)$ where l and r are two obstacle edges. Let h be the highest horizontal edge of E'_h (but no higher than the sweep line) which intersects both l and r. If no such edge exists, or if the region between h, l, r and the sweep line contains a point of $S \cup S'$, then I is a *non-ladder interval.* Otherwise I is a *ladder interval,* and its *current top rung* (denoted as *top*(I)) is defined to be h. It can be shown that if two ladder intervals are adjacent, then their current top rungs are the same.

We maintain the obstacle edges intersected by the sweep line as a balanced binary search tree T. The innovative idea used in our algorithm is that the additional properties of the intervals such as ladder/non-ladder classification and current top rung information, *are not maintained* in T. Instead, consider another tree T' which contains a subset of the obstacle edges intersected by the sweep line. Each interval M defined by these edges on the sweep line is either an original non-ladder interval, or is the union of a maximal contiguous sequence of original ladder intervals, called a *merged-ladder interval.* Each merged-ladder interval is associated with a horizontal edge of E'_h called *top*(M). The above information is maintained with each interval of T'.

We update T and T' as follows. Consider the event where the sweep line encounters an edge of E'_h, say $h = uv$. We search for u and v in T and find the intervals $I_u = (l_u, r_u)$ and $I_v = (l_v, r_v)$. We search for u and v in T' and find the intervals $M_u = (L_u, R_u)$ and $M_v = (L_v, R_v)$. If M_u is a merged-ladder, we classify I_u as ladder, otherwise as non-ladder. We similarly classify I_v. If I_u is ladder, we output the part of *top*(M_u) between the obstacle edges l_u and r_u. Similarly if I_v is ladder, we output the part of *top*(M_v) between the obstacle edges l_v and r_v. Let the sequence of edges in T' be $\alpha L_u R_u \beta L_v R_v \gamma$ where α, β and γ represent subsequences of edges. Using a constant number of balanced binary search tree operations (search/insert/delete/split/merge), we can achieve a new tree T' which has the sequence $\alpha L_u l_u r_u l_v r_v R_v \gamma$. In this tree, the intervals (L_u, l_u) and (r_v, R_v), if non-zero, are classified as merged-ladder with current top rungs the same as *top*(M_u) and *top*(M_v) respectively. The intervals (l_u, r_u) and (l_v, r_v) are classified as non-ladder. Finally, the interval (r_u, l_v), if non-zero, is classified as merge-ladder with current top rung h.

In this version we omit discussing how other events are handled (e.g. encountering top/bottom end points of obstacle edges), but claim that each requires at most a constant number of balanced binary search tree operations. Since each tree operation takes $O(\log n)$ time, the sweep takes $O(n \log n)$ time. Thus, the overall algorithm for constructing the Steiner spanner takes $O(n \log n + n/\epsilon^2)$ time.

4 All-pairs short and shortest paths in planar graphs

Let $G = (V(G), E(G))$ be a graph with nonnegative real edge-weights. The *all-pairs shortest paths* (APSP) problem asks for finding shortest paths between every pair of vertices in G, while the *single-source shortest paths* (SSSP) (or *shortest path tree*) problem asks for shortest paths between a specific vertex and all other vertices in G. For $s, t \in V(G)$, we will call the length of a shortest s-t path in G the *distance* between them, and the length of a c-short s-t path in G their *c-approximate distance.* For a subgraph H of G and vertices $s, t \in V(H)$, we will denote the distance from s to t in H by $\delta_H(s, t)$. For $s, t \in V(G)$, we will denote their distance in G simply by $\delta(s, t)$. A *separator* S_G of an n-vertex planar graph G is a set of vertices whose removal divides G into two subgraphs G_1 and G_2

such that $|V(G_i)| \leq 2n/3$, for $i = 1, 2$, no vertex of G_1 is adjacent to any vertex in G_2, and $|S_G| = O(\sqrt{n})$. Such a separator can be found in $O(n)$ time [24].

In this section we shall give a family of algorithms for solving the APSP and the all-pairs 2-short paths problems on an n-vertex undirected planar graph G_P with nonnegative real edge-weights. (*Remark:* All of our results for the APSP problem hold also for directed planar graphs with nonnegative real edge-weights.) Our algorithms first preprocess G_P (by creating a data structure) and then query this data structure to find a shortest or 2-short path between a query pair of vertices. Due to space limitations, we will only describe how distance queries are answered. The corresponding shortest (or short) path queries can be answered in an additional time proportional to the number of edges of the reported path.

We start with the APSP problem. We first sketch a simple algorithm, called BASIC-APSP, which provides the basis for the other algorithms in this section. This algorithm is a straightforward application of the divide-and-conquer technique using separators. (Although we have not found this algorithm in the literature, we suspect that it was known as a folklore.) Its preprocessing procedure consists of the following steps: (1) Perform a recursive separator decomposition of G_P and associate with it a binary tree called the *separator decomposition tree* $T(G_P)$. Moreover, associate with every node x of $T(G_P)$ a certain subgraph G of G_P, denoted by $x(G)$, and the separator S_G of this subgraph, denoted by $x(S_G)$. (The root is associated with the input graph G_P and its separator S_{G_P}.) If x is a leaf of $T(G_P)$, then consider all vertices of $x(G)$ as separator vertices. (2) For each node x of $T(G_P)$ and for every vertex $v \in x(S_G)$, compute a shortest path tree in $x(G)$ rooted at v, by running the $O(n)$-time SSSP algorithm of [23]. (3) Preprocess $T(G_P)$ in linear time (using the algorithm of [25]) such that lowest common ancestor (LCA) queries are answered in $O(1)$ time.

It is not hard to see that the running time is dominated by the running time, say $P(n)$, of Step (2) which satisfies the recurrence $P(n) \leq \max\{P(n_1)+P(n_2) : n_1+n_2 = n \text{ and } n_1, n_2 \leq 2n/3\}+O(n^{3/2})$, and whose solution is $P(n) = O(n^{3/2})$. Let x be a node of $T(G_P)$. For an ancestor (resp. descendant) node y of x in $T(G_P)$, we will call the subgraph $y(G)$ the *ancestor* (resp. *descendant*) subgraph of $x(G)$.

The main idea behind the query procedure is the following. Let $s, t \in V(G_P)$ be any two vertices and let x be the node of $T(G_P)$ for which the separator $x(S_G)$ separates s from t in the descendant subgraph $x(G)$ of G_P. Then clearly, $\delta_{x(G)}(s,t) = \min_{v \in x(S_G)}\{\delta_{x(G)}(s,v) + \delta_{x(G)}(v,t)\}$. However, it is possible that $\delta_{x(G)}(s,t) \neq \delta(s,t)$, since a shortest s-t path in G_P need not necessarily stay inside $x(G)$. For this reason we look for shortest s-t paths in the ancestor subgraphs of $x(G)$. The crucial observation is that in such a case the shortest path has to pass through some separation vertex of these subgraphs. Let $A(x) = \{y : y \text{ is an ancestor of } x \text{ in } T(G_P)\}$. Then, it is not hard to verify that $\delta(s,t) = \min\{\delta_{x(G)}(s,t), \min_{y \in A(x)}\{\min_{v \in y(S_G)}\{\delta_{y(G)}(s,v) + \delta_{y(G)}(v,t)\}\}\}$.

The query procedure of algorithm BASIC-APSP consists of the following steps. (1) Let y and z be the lowest-level nodes (i.e. closest to the root) of $T(G_P)$ such that $s \in y(G)$, $t \in z(G)$ and $y \neq z$. Find the LCA x of y and z in

$T(G_P)$. (2) Compute $\delta_{x(G)}(s,t)$ (as shown above). Set $\delta(s,t) = \delta_{x(G)}(s,t)$. If x is the root of T_G, then stop. (3) Let u be the parent of x. Compute $\delta(s,t) = \min\{\delta(s,t), \min_{v \in u(S_G)}\{\delta_{u(G)}(s,v) + \delta_{u(G)}(v,t)\}\}$. (4) If u is the root of $T(G_P)$, then stop. Otherwise, set $x = u$ and repeat Steps (3) and (4).

Clearly, Step (1) takes $O(1)$ time and Step (2) takes time $O(|x(S_G)|) = O(\sqrt{|V(x(G))|}) = O(\sqrt{n})$. Let $Q(i)$ be the maximum time required by Steps (3) and (4), where $0 \leq i \leq d$ is the level number of node x in $T(G_P)$ and $d = O(\log n)$ is the depth of $T(G_P)$. One iteration of Step (3) takes $O(((\frac{2}{3})^i n)^{1/2})$ time, since $|V(x(G))| = O((\frac{2}{3})^i n)$. Hence, $Q(i)$ satisfies the recurrence $Q(i) \leq Q(i-1) + O(((\frac{2}{3})^i n)^{1/2})$, whose solution is $O(\sqrt{n})$.

The above discussion implies that algorithm BASIC-APSP can answer a distance query between any two vertices in $O(\sqrt{n})$ time, after an $O(n^{3/2})$ time and space preprocessing of G_P. We now show how we can improve more on the query time. Frederickson [17] showed how to divide an n-vertex planar graph into $\Theta(n/r)$ edge-disjoint subgraphs, called *regions*, such that each region has $O(r)$ vertices, $1 \leq r \leq n$, and $O(\sqrt{r})$ *boundary* vertices (i.e. vertices shared with other regions). Such a division is called an *r-division* and can be computed in $O(n \log n)$ time [17], or even in $O(n)$ time using the results of Goodrich [19].

We call our new algorithm IMPROVED-APSP. Preprocessing procedure of algorithm IMPROVED-APSP: (P1) Find an r-division D of G_P. (P2) Run the preprocessing procedure of algorithm BASIC-APSP inside every region. (P3) For every boundary vertex b of D, compute a shortest path tree in G_P rooted at b using the algorithm of [23].

Query procedure of algorithm IMPROVED-APSP: Let $s,t \in V(G_P)$. (Q1) If s and t belong to the same region R, then find their distance $\delta_R(s,t)$ inside R (using the query procedure of algorithm BASIC-APSP), find $\delta'(s,t) = \min_{v \in B(R)}\{\delta(s,v) + \delta(v,t)\}$, (where $B(R)$ is the set of boundary vertices of R), and output as $\delta(s,t)$ the minimum of $\delta_R(s,t)$ and $\delta'(s,t)$. (Q2) If s and t belong to different regions R and R' respectively, then $\delta(s,t) = \min_{v \in B(R)}\{\delta(s,v)+\delta(v,t)\}$.

It can be easily verified that the preprocessing procedure of algorithm IMPROVED-APSP needs $O(n^2/\sqrt{r})$ time and space, while the query procedure of the same algorithm takes $O(\sqrt{r})$ time. This yields the first graph-theoretic result (I) mentioned in the introduction.

Both previous algorithms compute (exact) APSP information in planar graphs. We will now see how 2-short paths can be efficiently computed. Let s be a vertex of a planar graph G, and let $b_s \in S$, where S is a separator of G. The vertex b_s, satisfying that $\delta(s,b_s) \leq \delta(s,u)$, $\forall u \in S$, is called the *closest separator vertex* of s on the separator S. Our algorithm is based on the following lemma.

Lemma 6. *Let s and t be two vertices of a planar graph G and let S be a separator of G. If a shortest s-t path contains a vertex from S, then* $\min\{\delta(s,b_s)+\delta(b_s,t), \delta(t,b_t)+\delta(b_t,s)\} \leq 2\delta(s,t)$.

Proof. Consider a shortest s-t path P in G. Let u be the first vertex of S in P. Then, the length of P is given by $\delta(s,t) = \delta(s,u) + \delta(u,t)$. Consider first the case that $\delta(s,u) \leq \delta(u,t)$. Since $\delta(s,b_s) \leq \delta(s,u)$ by the definition of b_s, we have

$\delta(b_s,t) \le \delta(b_s,u)+\delta(u,t) \le \delta(b_s,s)+\delta(s,u)+\delta(u,t) \le 2\delta(s,u)+\delta(u,t)$. Thus, $\delta(s,b_s)+\delta(b_s,t) \le \delta(s,u)+2\delta(s,u)+\delta(u,t) \le 2\delta(s,u)+2\delta(u,t) = 2(\delta(s,u)+\delta(u,t)) = 2\delta(s,t)$, where the second inequality follows from our assumption that $\delta(s,u) \le \delta(u,t)$. In the other case, namely when $\delta(u,t) \le \delta(s,u)$, we can show in a similar way that $\delta(t,b_t)+\delta(b_t,s) \le 2\delta(s,t)$. □

We now present algorithm 2-APPROX-APSP that computes all-pairs 2-short paths in G_P. The preprocessing procedure is the same as that of algorithm BASIC-APSP, except for the computation of the closest separator vertices. However, observe that the required closest separator vertices, for every $u \in x(G)$, can be easily found in Step (2) (within the same resource bounds) from the shortest path trees rooted at every vertex $v \in x(S_G)$.

The query procedure of algorithm 2-APPROX-APSP is based on Lemma 6, and its structure is similar to that of algorithm BASIC-APSP. Let $s,t \in V(G_P)$ be the query vertices. The query procedure is as follows: (Q1) Let y and z be the lowest-level nodes (i.e. closest to the root) of $T(G_P)$ such that $s \in y(G)$, $t \in z(G)$ and $y \neq z$. Find the LCA x of y and z in $T(G_P)$. (Q2) Compute $\delta'_{x(G)}(s,t) = \min\{\delta_{x(G)}(s,b_s)+\delta_{x(G)}(b_s,t), \delta_{x(G)}(t,b_t)+\delta_{x(G)}(b_t,s)\}$, where b_s (resp. b_t) denotes the closest separator vertex of s (resp. of t) on the separator $x(S_G)$. Set $\delta 2(s,t) = \delta'_{x(G)}(s,t)$. If x is the root of $T(G_P)$, then stop. (Q3) Let u be the parent of x. Compute $\delta'_{u(G)}(s,t) = \min\{\delta_{u(G)}(s,b_s)+\delta_{u(G)}(b_s,t), \delta_{u(G)}(t,b_t)+\delta_{u(G)}(b_t,s)\}$, where now b_s (resp. b_t) denotes the closest separator vertex of s (resp. of t) on the separator $u(S_G)$. Set $\delta 2(s,t) = \min\{\delta 2(s,t), \delta'_{u(G)}(s,t)\}$. (Q4) If u is the root of $T(G_P)$, then stop. Otherwise, set $x = u$ and repeat Steps (3) and (4).

The correctness of the query procedure is established by the following lemma.

Lemma 7. *Let s,t be two vertices in G_P, and let y,z,x and $\delta'_{x(G)}(s,t)$ be as defined above. Then,* $\min\{\delta'_{x(G)}(s,t), \min\{\delta_{w(G)}(s,b_s)+\delta_{w(G)}(b_s,t), \delta_{w(G)}(t,b_t)+\delta_{w(G)}(b_t,s)\}\} \le 2\delta_{G_P}(s,t)$ *for some ancestor w of x in $T(G_P)$, where b_s (resp. b_t) is the closest separator vertex of s (resp. t) on the separator $w(S_G)$ in the subgraph $w(G)$.*

Proof. Omitted due to space limitations. □

Concerning the resource bounds of the query procedure, note that Steps (Q1) and (Q2) take $O(1)$ time. As before, let $Q(i)$ be the maximum time required by Steps (Q3) and (Q4), where $0 \le i \le d$ is the level number of node x in $T(G_P)$ and $d = O(\log n)$ is the depth of $T(G_P)$. One iteration of Step (Q3) clearly takes $O(1)$ time, since the required distances are available from the preprocessing of the graph. Hence, $Q(i)$ satisfies the recurrence $Q(i) \le Q(i-1)+O(1)$, whose solution is $O(\log n)$. Therefore, algorithm 2-APPROX-APSP answers a length query between any two vertices of G_P in $O(\log n)$ time, after an $O(n^{3/2})$ time and space preprocessing, yielding the second graph-theoretic result (II) mentioned in the introduction.

5 Short path queries among obstacles in the plane

Following Chen's approach for processing L_2 short path queries [5], our L_p short path data structures consist of two major components:

Part A. A data structure for answering queries on all-pairs short(est) paths in a graph G_P which is the planar Steiner L_p τ-spanner constructed by our algorithm in Section 3.

Part B. A data structure that, given any two query points s and t in the plane, quickly reduces the computation of a short s-t path to computing the short(est) paths between a constant number of vertices of G_P.

Given the results in Sections 3 and 4, the data structure of Part A can be easily constructed as follows. Let $\epsilon_1 > 0$ be a value depending on the required stretch factor. We first obtain a planar Steiner L_p $(2^{(p-1)/p} + \epsilon_1)$-spanner with $O(n)$ Steiner vertices, denoted by G_P, and then build a data structure for answering all-pairs short(est) path queries in the planar graph G_P. It is clear from Section 4 that, for the results of the first and third row of Table 1, an all-pairs short(est) path query data structure on G_P can be built in the claimed time and space bounds. It should be also clear that the result of the second row can be achieved by applying to G_P the linear time SSSP algorithm of [23]. The result of the fourth row follows by applying to G_P Chen's all-pairs short path query algorithm on planar graphs [5].

The data structure of Part B is the same for all results of Table 1, and is a generalization of Chen's approach [5] for processing L_2 short path queries. (We omit the details due to space limitations.) The data structure of Part B can be set up in $O((n \log n)/\epsilon_2)$ time and $O(n/\epsilon_2)$ space, for some $\epsilon_2 > 0$ depending on the required stretch factor. Given two query points s and t in the plane, the data structures of Parts A and B together enable us to compute the length of a geometric short s-t path in $O((\log n)/\epsilon_2)$ time from the "graphic" short(est) paths between $O((1/\epsilon_2)^2)$ pairs of vertices in G_P.

6 Final Remarks

We have recently improved the planar spanner results presented in this paper using new ideas. More precisely, we can achieve a planar $(1+\epsilon)$-spanner with a linear number of Steiner vertices for *all* metrics (i.e., the constant c, in Table 1, is equal to 1 for any L_p-metric). The details will be given in the full paper.

References

1. S. Arya, G. Das, D.M. Mount, J.S. Salowe and M. Smid, "Euclidean spanners: short, thin, and lanky", *Proc. 27th ACM STOC*, 1995, pp. 489-498.
2. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A. Wu, "An optimal algorithm for approximate nearest neighbor searching", *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994, pp. 573–582.
3. M. J. Atallah and D. Z. Chen, "Parallel rectilinear shortest paths with rectangular obstacles," *Comp. Geometry: Theory and Appl.*, 1:2 (1991), pp.79-113.

4. M. J. Atallah and D. Z. Chen, "On parallel rectilinear obstacle-avoiding paths," *Computational Geometry: Theory and Applications*, 3 (1993), pp. 307–313.
5. D. Z. Chen. "On the all-pairs Euclidean short path problem," *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, 1995, pp. 292–301.
6. D. Z. Chen and K. S. Klenk, "Rectilinear short path queries among rectangular obstacles," *Proc. 7th Can. Conf. on Comp. Geometry*, 1995, pp. 169–174.
7. D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu, "Rectilinear shortest path queries among weighted obstacles in the rectilinear plane," *Proc. 11th Annual ACM Symp. on Computational Geometry*, 1995, pp. 370–379.
8. L. P. Chew, "Constrained Delaunay triangulations," *Algorithmica*, 4 (1989), pp. 97–108.
9. L. P. Chew, "There are planar graphs almost as good as the complete graph," *J. of Computer and System Sciences*, 39 (1989), pp. 205–219.
10. L. P. Chew, "Planar graphs and sparse graphs for efficient motion planning in the plane," Computer Science Tech Report, PCS-TR90-146, Dartmouth College.
11. L. P. Chew and R. L. Drysdale, "Voronoi diagrams based on convex distance functions," *Proc. 1st Annual ACM Symp. on Comp. Geometry*, 1985, pp. 235–244.
12. Y.-J. Chiang, F. P. Preparata, and R. Tamassia, "A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps," *Proc. of the 4th ACM-SIAM Symp. on Discrete Algorithms*, 1993, pp. 44–53.
13. K. L. Clarkson, "Approximation algorithms for shortest path motion planning," *Proc. 19th Annual ACM Symp. Theory of Computing*, 1987, pp. 56–65.
14. H. Djidjev, G. Pantziou and C. Zaroliagis, "On-line and dynamic algorithms for shortest path problems", *Proc. 12th Symp. on Theor. Aspects of Comp. Sc.*, LNCS 900, Springer-Verlag, 1995, pp. 193-204.
15. D. P. Dobkin, S. J. Friedman, and K. J. Supowit, "Delaunay graphs are almost as good as complete graphs," *Discrete & Comp. Geometry*, 5 (1990), pp. 399–407.
16. H. ElGindy and P. Mitra, "Orthogonal shortest route queries among axes parallel rectangular obstacles," *Int. J. of Comp. Geometry and Appl.*, 4 (1) (1994), 3–24.
17. G. Frederickson, "Fast algorithms for shortest paths in planar graphs, with applications", *SIAM J. on Computing*, 16 (1987), pp.1004-1022.
18. G.N. Frederickson, "Using cellular graph embeddings in solving all pairs shortest path problems", *J. of Algorithms*, 19 (1995), pp. 45-85.
19. M. Goodrich, "Planar separators and parallel polygon triangulation", *Proc. 24th ACM Symp. on Theory of Comp.*, 1992, pp.507-516.
20. M.T. Goodrich and R. Tamassia, "Dynamic ray shooting and shortest paths via balanced geodesic triangulations," *Proc. 9th Annual ACM Symp. on Computational Geometry*, 1993, pp. 318–327.
21. L.J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," *Proc. 3rd Annual ACM Symp. on Computational Geometry*, 1987, pp. 50–63.
22. L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan, "Linear time algorithms for visibility and shortest paths problems inside triangulated simple polygons," *Algorithmica*, 2 (1987), pp. 209–233.
23. P. Klein, S. Rao, M. Rauch and S. Subramanian, "Faster shortest-path algorithms for planar graphs", *Proc. 26th ACM Symp. on Theory of Comp.*, 1994, pp.27-37.
24. R. Lipton and R. Tarjan, "A separator theorem for planar graphs", *SIAM J. Applied Mathematics*, 36 (1979), pp.177-189.
25. B. Schieber and U. Vishkin, "On finding lowest common ancestors: Simplification and parallelization", *SIAM J. Computing*, 17:6 (1988), pp.1253-1262.