SIAM J. COMPUT.
Vol. 39, No. 8, pp. 3814–3829

LINEAR-TIME ALGORITHMS FOR GEOMETRIC GRAPHS WITH SUBLINEARLY MANY EDGE CROSSINGS*

DAVID EPPSTEIN†, MICHAEL T. GOODRICH†, AND DARREN STRASH†

Abstract. We provide linear-time algorithms for geometric graphs with sublinearly many edge crossings. That is, we provide algorithms running in $O(n)$ time on connected geometric graphs having n vertices and k pairwise crossings, where k is smaller than n by an iterated logarithmic factor. Specific problems that we study include Voronoi diagrams and single-source shortest paths. Our algorithms all run in linear time in the standard comparison-based computational model; hence, we make no assumptions about the distribution or bit complexities of edge weights, nor do we utilize unusual bit-level operations on memory words. Instead, our algorithms are based on a *planarization* method that "zeros in" on edge crossings, together with methods for applying planar separator decompositions to geometric graphs with sublinearly many crossings. Incidentally, our planarization algorithm also solves an open computational geometry problem of Chazelle for triangulating a self-intersecting polygonal chain having n segments and k crossings in linear time, for the case when k is sublinear in n by an iterated logarithmic factor.

Key words. geometric graphs, Voronoi diagrams, epsilon-cuttings, trapezoidal maps, arrangements, shortest paths

AMS subject classifications. 68W20, 68W40, 52C30, 05C85, 05C10

DOI. 10.1137/090759112

1. Introduction.

A *geometric graph* [50] is an embedding of a graph $G = (V, E)$ in $\mathbf{R}^2$ so that each vertex v is associated with a unique point p in $\mathbf{R}^2$ and each edge is "drawn" as a straight line segment joining the points associated with its end vertices. Moreover, the edges incident on each vertex v are given in angular order around v, so that faces in the embedding of G in $\mathbf{R}^2$ are well defined (e.g., using the next-clockwise-edge ordering). Thus, we use the same notation and terminology to refer to G and its embedding. If the edges in G have no crossings, then G is said to be a *plane graph*, while graphs that admit realizations as plane graphs are *planar graphs* [20, 29].

Geometric graphs are natural abstractions of the geometric and connectivity relationships that arise in a number of applications, including road networks, railroad networks, and utility distribution grids, as well as sewer lines and the physical connections defining the Internet. An example road network is shown in Figure 1.1.

Although planar graphs and their plane graph realizations have been studied extensively (e.g., see [57]), real-world geometric graphs often contain edge crossings. Recent experimental studies by the first two authors give empirical evidence that real-world road networks typically have $\Theta(\sqrt{n})$ edge crossings, where n is the number of vertices [25]. Motivated by this real-world example, therefore, we are interested in studying algorithms for connected geometric graphs that have a sublinear number of edge crossings. However, we use a weaker restriction on the number of crossings than the bounds that our evidence suggests for road networks: here we are interested in

*Received by the editors May 14, 2009; accepted for publication (in revised form) September 10, 2010; published electronically December 14, 2010. This research was supported in part by the National Science Foundation, under grants 0724806, 0713046, and 0830403, and by the Office of Naval Research, under MURI award N00014-08-1-1015. A preliminary version of this paper appeared in Proceedings of the the ACM-SIAM Symposium on Discrete Algorithms (SODA) as [26].
http://www.siam.org/journals/sicomp/39-8/75911.html
†Department of Computer Science, University of California, Irvine, Irvine, CA 92697-3425 (eppstein@ics.uci.edu, goodrich@ics.uci.edu, dstrash@ics.uci.edu).

FIG. 1.1. *A portion of the road network for Manhattan, New York. This image is from* http://wiki.openstreetmap.org/, *under the Creative Commons attribution–share alike license.*

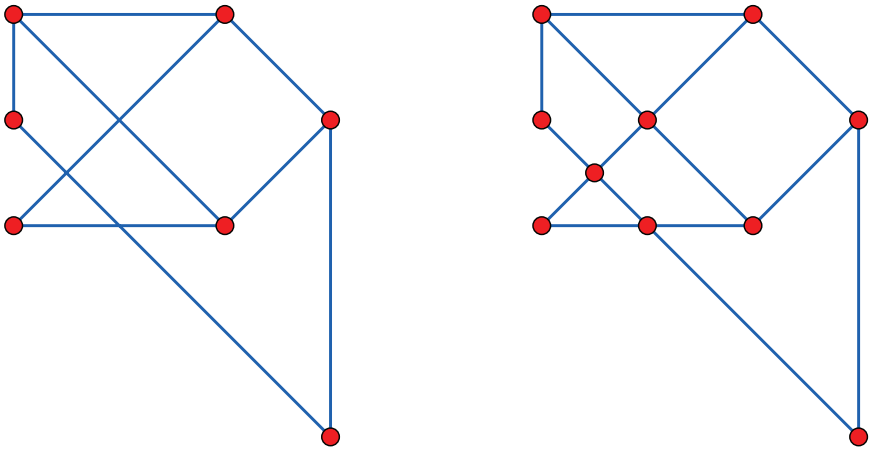

FIG. 1.2. *A geometric graph and its planarization.*

n-vertex geometric graphs that have at most $O(n/\log^{(c)} n)$ pairwise edge crossings, for some constant c, where $\log^{(c)} n$ denotes the cth iterated logarithm function. We allow more than two edges to cross at the same point; however, these crossings are counted as the number of pairwise edges involved. We refer to such geometric graphs as *restrained* graphs.

Given an n-vertex geometric graph G, the *planarization*[1] of G is the graph G' that is defined by the arrangement of the edges in G. That is, as shown in Figure 1.2, we place a vertex in G' for every vertex and pairwise edge crossing in G, and we

[1]Our use of this term differs from its use in the graph drawing literature (e.g., see [20]), where it refers to the problem of removing a minimal number of edges to make G be planar.

create an edge in G' for every maximal edge segment from G that connects exactly two vertices in G'. Likewise, we preserve the (clockwise/counterclockwise) ordering of edges around corresponding vertices in G and G', and we assume that intersection vertices in G' similarly have their edges given in rotational order. Thus, G' is a plane graph having $n+k$ vertices, where k is the number of pairwise edge crossings among the edges in G. By well-known properties of planar graphs (e.g., see [49, Prop. 2.1.6]), this implies that G' has at most $3n+3k-6$ edges, which in turn implies that G has at most $3n+k-6$ edges. Therefore, by restricting our attention to connected geometric graphs with a sublinear number of edge crossings, we are, by implication, focusing on connected geometric graphs that have $O(n)$ edges in their planarizations.

As mentioned above, a wealth of algorithms are known for planar graphs and plane graphs. Indeed, many of these algorithms, for such problems as single-source shortest paths and minimum spanning trees, run in $O(n)$ time. Much less is known for nonplanar geometric graphs, however, which motivates our interest in such graphs in this paper. Specifically, we are interested in the following problems for connected, restrained geometric graphs:

- The *Voronoi diagram* problem, which is also known as the *post office* problem: we are given a set P of k vertices in a geometric graph G and asked to determine for every other vertex v in G the vertex in P that is closest to v according to the graph metric.
- The *single-source shortest path* problem: we are given a vertex s and a geometric graph G and asked to find the shortest paths from s to every other vertex in G.
- The *polygon planarization* problem: given a geometric graph defining a nonsimple polygon P having n vertices, compute the arrangement of all the edges of P, including vertices defined by the pairwise crossings of the edges in P.

In all these cases, we desire comparison-based algorithms that require no additional assumptions regarding the distribution of edge weights, so that our algorithms can apply to a wide variety of possible edge weights that may vary for different users, including combinations of distance, travel time, toll charges, and subjective scores rating safety and scenic interest [23].

1.1. Previous related work. In the algorithms community, there has been considerable prior work on shortest path algorithms for Euclidean graphs (e.g., see [32, 38, 40, 53, 54, 58]), which are geometric graphs where edges are weighted by the lengths of the corresponding line segments. This prior work takes a decidedly different approach than we take in this paper, however, in that it focuses on using special properties of the edge weights that do not hold in the comparison model, whereas we study road networks as geometric graphs with a sublinear number of edge crossings and we desire linear-time algorithms that hold in the comparison model.

The specific problems for which we provide linear-time algorithms are well known in the general algorithms and computational geometry literatures. For general graphs with n vertices and m edges, excellent work can be found on efficient algorithms in the comparison model, including single-source shortest paths [16, 34, 51], which can be found in $O(n \log n + m)$ time [30], and Voronoi diagrams [4, 5], whose graph-theoretic version can be constructed in $O(n \log n + m)$ time [28, 44]. None of these algorithms run in linear time, even for planar graphs. Linear-time algorithms for planar graphs are known for single-source shortest paths [37], but these unfortunately do not immediately translate into linear-time algorithms for nonplanar geometric graphs.

A number of efficient algorithms have been developed for the RAM model of computation under various edge weight restrictions. For example, if edge weights are restricted to be positive integers, the algorithm of Thorup [56] runs in linear time. If negative integers are allowed, then Goldberg's algorithm [31] takes $O(\sqrt{n}m \log N)$ time, where N is the absolute value of the smallest negative integer edge weight. Furthermore, if edge weights are selected at random in the range $[0, 1]$, then the algorithm of Meyer [45] achieves linear average-case time complexity. However, the methods used in these algorithms exclude them from being applicable in the comparison model.

Chazelle [8] shows that any simple polygon can be triangulated in $O(n)$ time and that this algorithm can be extended to determine in $O(n)$ time, for any polygonal chain P, whether or not P contains a self-intersection. In addition, Chazelle posed as an open problem whether or not one can compute the arrangement of a nonsimple polygon in $O(n+k)$ time, where k is the number of pairwise edge crossings. Clarkson, Cole, and Tarjan [14, 13] answer this question in the affirmative for polygons with a superlinear number of crossings, as they give a randomized algorithm that solves this problem in $O(n \log^* n + k)$ expected time. There is, to our knowledge, no previous algorithm that solves Chazelle's open problem, however, for nonsimple polygons with a sublinear number of edge crossings.

1.2. Our results. In this paper, we provide the first linear-time algorithm for planarizing a nonplanar connected geometric graph having a number of pairwise edge crossings, k, that is sublinear in the number of vertices, n, by an iterated logarithmic factor. Specifically, we provide a randomized algorithm for planarizing geometric graphs in $O(n + k \log^{(c)} n)$ expected time, which is linear for restrained geometric graphs. Given such a planarization, we show how to apply $O(\sqrt{n})$-separator decompositions to produce linear-time algorithms for a number of problems on restrained geometric graphs, including Voronoi diagrams and single-source shortest paths. We also show how our planarization algorithm can be used to solve Chazelle's open problem of planarizing nonsimple polygons in expected linear time for polygons having a number of pairwise edge crossings that is sublinear in n by an iterated logarithmic factor. Thus, combining this result with the polygon planarization algorithm of Clarkson, Cole, and Tarjan [14, 13] provides a method for planarizing an n-vertex polygon with k edge crossings in optimal $O(n + k)$ expected time, for all values of k except those in the range $[n/\log^{(c)} n, n \log^* n]$. Our result also implies that the convex hull of restrained nonsimple polygons can be constructed in $O(n)$ expected time, which, to the best of our knowledge, was also previously open.

Besides planar separator decompositions, which we discuss below, another one of the techniques used in this paper is a method for constructing a $(1/r)$-cutting for the edges of a geometric graph, G. This is a proper triangulation,[2] T, of the interior of the bounding box containing G such that any triangle t in T intersects at most $(1/r)n$ edges of G. Using existing methods (e.g., see [1, 17, 36]), one can construct such a $(1/r)$-cutting for G in $O(n \log r + (r/n)k)$ time, where n is the number of vertices in G and k is the number of pairwise edge crossings. However, in our application such a bound would be nonlinear, as we require r to be large. We show, in section 4, that for connected geometric graphs such a cutting can be constructed in the faster expected time bound $O(ns + (r/n)k)$, where $r \leq n/\log^{(s)} n$.

2. Separator decompositions. One of the main ingredients in our algorithms is the existence of small separators in certain graph families (e.g., see [42, 46]). Several

[2] A *proper triangulation* is a connected planar geometric graph such that every face is a triangle and every triangular face has exactly three vertices on its boundary.

of the algorithms in this paper are based on the use of separators: we use them both as part of our algorithm for finding cuttings of geometric graphs and later, once the graph has been planarized. Hence, we briefly review these tools here.

Given a graph $G = (V, E)$, a subset W of V is an $f(n)$-*separator* if the removal of the at most $f(n)$ vertices in W separates G into two subgraphs G_1 and G_2, each containing at most δn vertices, for some constant $0 < \delta < 1$. It is well known that planar graphs have $O(\sqrt{n})$-separators with $\delta = 2/3$, and that such separators can be constructed in $O(n)$ time [42]. Such separators are typically used in divide-and-conquer algorithms, which involve finding a separator, recursively solving the problem in the two separated subgraphs, and then merging the solutions together. If the merge and divide steps can be solved in $o(n)$ time, however, it is useful to have the entire recursive separator decomposition computed in advance, for otherwise there is no way to beat an $O(n \log n)$ time bound. Such a *separator decomposition* defines a binary tree B such that the root of B is associated with the $f(n)$-separator for G and the subtrees of this root are defined recursively for the graphs G_1 and G_2.

Previous work on separators includes the seminal contribution of Lipton and Tarjan [42], who show that $O(\sqrt{n})$-sized separators exist for n-vertex planar graphs and that these can be computed in $O(n)$ time. Goodrich [33] shows that recursive $O(\sqrt{n})$-separator decompositions can be constructed for planar graphs in $O(n)$ time. A related concept is that of geometric separators, which use geometric objects to define separators in graphs defined by systems of intersecting disks (e.g., see [3, 48, 47, 55]). Eppstein, Miller, and Teng [27] provide a linear-time construction algorithm for geometric separators which translates into an $O(n \log n)$ recursive separator decomposition algorithm.

Because restrained graphs are not planar, the result of Goodrich does not immediately apply. However, it can be applied once we have planarized the graph, and it can also be applied to planar structures formed from subsets of the graph, such as the one we describe in the next section.

3. Trapezoidal decomposition of a sample. Suppose we are given a geometric graph G having n vertices and k pairwise intersections among its edges. In this section, we describe our algorithm for constructing a trapezoidal decomposition of a random sample of the edges of G. That is, given a sample of r edges, we construct the arrangement of these edges together with a set of vertical line segments through each edge endpoint and crossing, where each such segment is maximal with respect to the property of not crossing any other sampled edge, as shown in Figure 3.1. Our method is parameterized by s, where $r \leq n/\log^{(s)} n$, and the sample probability is inversely proportional to $\log^{(s)} n$. We will later show how to refine this sample so that we can produce a cutting and then a planarization of G.

This first step of our algorithm is essentially the same as performing s levels of the Clarkson, Cole, and Tarjan algorithm, except that their method is for polygonal chains, whereas ours is for geometric graphs. Thus, we describe it at a high level.

Our algorithm begins with a trivial trapezoidal decomposition T_0 containing a single trapezoid that encloses all of G. Call this trapezoid t. Let $C(t) = E$ be the *conflict list* for t, that is, the set of edges from G that intersect the interior of t. Then, for $i = 1$ to s, we perform the following computation:

1. Find a random sample S_i of size $n/\log^{(i)} n$ of the edges in G, and for each trapezoid t in T_{i-1}, use the Bentley–Ottmann algorithm [6] to construct the trapezoidal decomposition of the arrangement of the segments in $C(t) \cap S_i$. Once all these trapezoidal decompositions are constructed, merge them to-

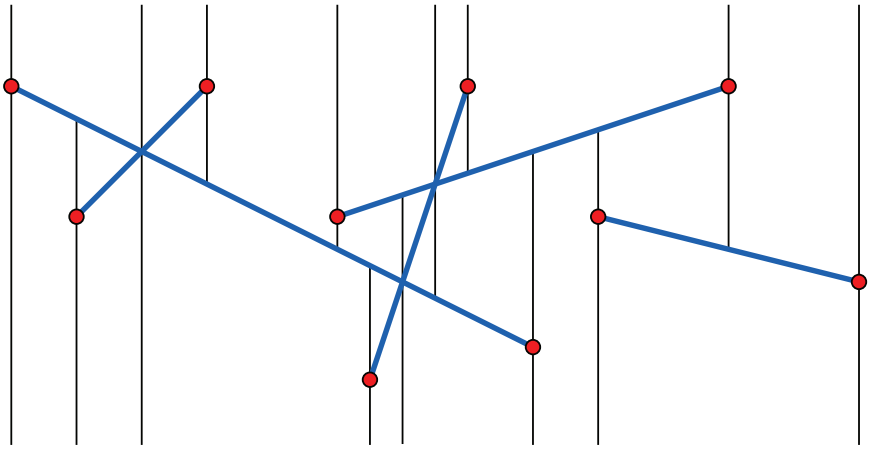

FIG. 3.1. *Trapezoidal decomposition of a sampled subset of input graph edges.*

gether to create a single trapezoidal decomposition, T_i, for the segments in S_i. To be consistent with Clarkson, Cole, and Tarjan, we choose the samples such that $S_1 \subset S_2 \subset \cdots \subset S_s$.

2. Perform a depth-first traversal of G, while keeping track of the trapezoids in the trapezoidal decomposition that are intersected during the walk, so as to determine, for each trapezoid t in T_i, the set $C(t)$. Since the geometric graph is connected, we never have to restart the depth-first traversal from a node whose location we do not already know. We can therefore use the arrangement of the sampled line segments to keep track of the intersected trapezoids at each step of the traversal. Thus we eliminate the need for time-consuming point-location data structure lookups.

Let $T = T_s$ be the resulting final trapezoidal decomposition that we get from this computation, and let $S = S_s$ be the final random sample. Using the framework established by Clarkson and Shor [15] for randomized divide-and-conquer algorithms such as this, we can show that

$$E(|T|) = O\left(r + \left(\frac{r}{n}\right)^2 k\right) \tag{3.1}$$

and

$$E\left(\sum_{t \in T} |C(t)|\right) = O\left(n + \left(\frac{r}{n}\right) k\right). \tag{3.2}$$

In particular, (3.1) is from their Lemma 4.1, and (3.2) follows from their Corollary 4.4. The number of steps in the depth-first traversal is proportional to the total size of the conflict lists of the input geometric graph with the trapezoidal decomposition, which as we have seen above is small. A step from one trapezoid to a horizontally adjacent trapezoid may be accomplished in constant time, but a single trapezoid may have a nonconstant number of neighbors above and below it, causing steps in those directions to take longer. However, as Clarkson, Cole, and Tarjan show, if $d(t)$ is the

number of trapezoids that neighbor trapezoid t, then

$$E\left(\sum_{t\in T}|C(t)|\cdot d(t)\right)=O\left(n+\left(\frac{r}{n}\right)k\right),$$

and this sum bounds the time to step vertically from one trapezoid to another using a sequential search along the trapezoid boundary to find the neighboring trapezoid. Therefore, we have the following preliminary result.

LEMMA 3.1. *Given a connected geometric graph G, with n edges and k pairwise edge crossings, and a parameter s, we can in expected time $O(ns+(r/n)k)$ find a random sample of $r=O(n/\log^{(s)} n)$ edges from G, the trapezoidal decomposition induced by the sample, and the set of edges of G crossing each trapezoid of the sample.*

4. Cuttings. At this stage we take a detour from the Clarkson, Cole, and Tarjan algorithm. For each trapezoid t in T, let $\alpha_t=|C(t)|r/n$. That is, α_t is the degree of excess that the conflict list for t has beyond what we would like for a $(1/r)$-cutting. For each trapezoid t with $\alpha_t>1$, we form a random sample, R_t, of $C(t)$ of size $2K_{\max}\alpha_t\log\alpha_t$, where $K_{\max}$ is the constant from Corollary 4.4 of Clarkson and Shor [15]. We then form the trapezoidal decomposition, T_t, of the arrangement of the segments in R_t using any quadratic-time line segment arrangement algorithm [2, 7, 10, 21]. Thus, by Corollary 4.4 from Clarkson and Shor [15], the maximum size of any conflict list of a trapezoid in T_t is expected to be less than

$$\begin{aligned}\left(\frac{|C(t)|}{|R_t|}\right)\log|R_t| &= \left(\frac{n}{r}\right)\left(\frac{1}{\log\alpha_t^2}\right)\log(2\alpha_t\log\alpha_t)\\ &\le \frac{n}{r}\end{aligned}$$

for $\alpha_t\ge 4$. Thus, we can repeat the above algorithm an expected constant number of times until we have this condition satisfied, which gives us one of the crucial properties of a $(1/r)$-cutting: namely, that each cell intersects at most (n/r) edges of G.

In addition, the number of new trapezoids created inside t, as well as the running time for creating the trapezoidal diagram T_t, is certainly at most $O(|R_t|^2)$, which is $O(\alpha_t^2\log^2\alpha_t)$. More importantly, we have the following result.

LEMMA 4.1. *Given the above construction applied to each trapezoid t in T, then*

$$E\left(\sum_{t\in T}\alpha_t^2\log^2\alpha_t\right)=O\left(r+\left(\frac{r}{n}\right)^2 k\right).$$

Proof. Our proof is based on an application of Theorem 3.6 from the Clarkson–Shor framework. To apply this theorem, we bound

$$E\left(\sum_{t\in T}\alpha_t^2\log^2\alpha_t\right)$$

by bounding the term $\alpha_t^2\log^2\alpha_t$ by

$$W\left(\binom{|C(t)|}{c}\right),$$

where W is a positive concave function on $\mathbf{R}^+$ and c is a constant. Here, for the sake of an upper bound, we take $c = 3$ and define

$$W(x) = \left(\frac{x^{1/3}}{N}\right)^2 \log^2 \frac{x^{1/3} + N}{N},$$

where $N = n/r$. Finally, to apply Theorem 3.6 from [15], we need to observe that the number of trapezoids in T that have a conflict list size at most c is proportional to the number of trapezoids in T that have a conflict list size at least 0, which is $|T|$. To see this, note that we can extend the vertical edges of any trapezoid in T in at most $O(1)$ ways until it hits $i = 1, 2, 3$ other edges of the random sample, S, at which point we can extend this trapezoid horizontally in $O(1)$ ways until we hit three segments in total. Therefore, by Theorem 3.6 from [15],

$$E\left(\sum_{t \in T} \alpha_t^2 \log^2 \alpha_t\right)$$

is

$$O\left(r + \left(\frac{r}{n}\right)^2 k\right). \qquad \square$$

Thus, our refined trapezoidal decomposition, T', will have size proportional to $|T|$. It is still not quite a $(1/r)$-cutting, however, as it is not a proper triangulation. Indeed, some trapezoids may have many more than four vertices on their boundaries (see Figure 4.1).

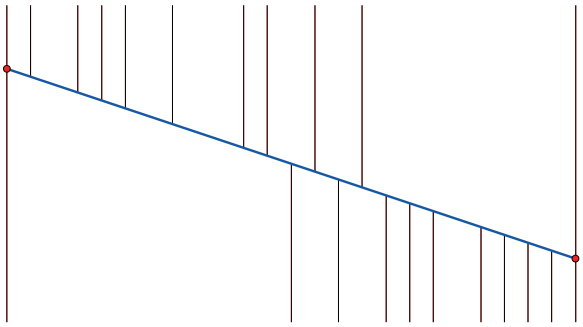

FIG. 4.1. *Many trapezoids may be adjacent to another trapezoid along its top or bottom edges.*

To refine T' into a proper triangulation, we borrow an idea from the fractional cascading framework of Chazelle and Guibas [11] to first refine T' into a trapezoidal decomposition such that each trapezoid has $O(1)$ vertices on its boundary, while keeping the total number of trapezoids to $O(|T'|)$, which is expected to be

$$O\left(r + \left(\frac{r}{n}\right)^2 k\right).$$

By triangulating the interior of each such trapezoid, we will get a $(1/r)$-cutting whose size is still $O(|T'|)$. (See Figure 4.2.)

Construct the graph-theoretic planar dual U to T', and note that we can direct the edges of U so as to define four directed-acyclic graphs, which respectively define

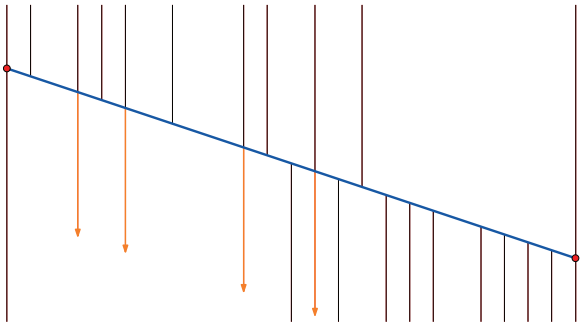

FIG. 4.2. *The cascading of trapezoidal rays.*

the partial orders "below," "above," "left-of," and "right-of" among the trapezoids. Without loss of generality, let us direct U according to the "below" relation, perform a topological sort, and process the trapezoids of T' from top to bottom according to this ordering. When processing a trapezoid, t, we assume inductively that we have determined the ordered list of vertices $V_t = (v_1, v_2, \ldots, v_j)$ on t's upper edge, which are bottom vertices of trapezoids above t. To process t we choose every other vertex, v_{2i}, in V_t and extend a vertical segment from v_{2i} to the bottom of t to split t in two for each such v_{2i}. Doing this for every other vertex in V_t, therefore, splits t and increases the number of trapezoids by $\lfloor |V_t|/2 \rfloor$. We then repeat this computation by considering the new set of trapezoids according to the "above" relation, from bottom to top. Next, we do a similar computation for the "left-of" and "right-of" relations (except that now we extend segments parallel to the top or bottom edges of our trapezoid in a way that partitions its interior into noncrossing trapezoids). When we have completed this last scan of the trapezoids, we will have created a trapezoidal decomposition such that each trapezoid has $O(1)$ vertices on its edges. More importantly, we also have the following result.

LEMMA 4.2. *The total number of trapezoids created by the above refinement process is $O(|T|)$, which has expected value $O(r + (r/n)^2 k)$.*

Proof. We have already established that $E(|T|)$ is $O(r+(r/n)^2 k)$ and that $E(|T'|)$ is $O(E(|T|))$. So we have yet to show that the number of new trapezoids created during any of our splitting processes is $O(|T'|)$. We do this by an accounting argument. Without loss of generality, consider the processing according to the "below" relation. Assume, for the sake of our analysis, that, at the beginning of our computation, we give each vertical edge in our trapezoidal decomposition \$2 and require every vertical edge at the end of the process to have at least \$1. When we extend a vertical ray from an even-numbered vertex v_{2i} at the top of a trapezoid t, we can assume inductively that the vertical edge above v_{2i} has \$2, as does the vertical edge directly to the left of this edge (which hits t at vertex v_{2i-1}). Let us take \$1 from this vertical edge and from the one that hits t at v_{2i}, which leaves \$1 at each of those edges, and use the \$2 to pay for the new vertical edge that we then extend through t. Therefore, since the two vertical edges we just took money from will not be processed again, we can process each trapezoid and pay for every action, while keeping \$1 for each trapezoid in our refined trapezoidal decomposition. Repeating this accounting argument for the "above," "left-of," and "right-of" relations completes the proof. ∎

Given a trapezoidal diagram having $O(1)$ vertices on the boundary of each trapezoid, and each trapezoid intersecting at most (n/r) edges of our geometric graph G,

we can easily triangulate each trapezoidal face in this diagram to turn it into a $(1/r)$-cutting with a number of triangles that is proportional to the number of trapezoids. (See Figure 4.3.)

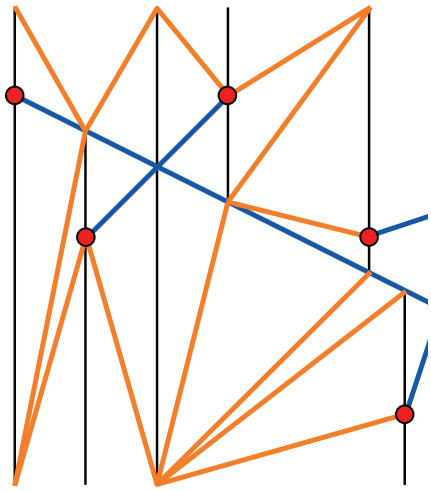

FIG. 4.3. *The triangulation step.*

Thus, putting all the pieces together, we get the following.

THEOREM 4.3. *Given a connected geometric graph G having n vertices and k pairwise edge crossings, one can construct a $(1/r)$-cutting for the edges of G of expected size $O(r + (r/n)^2 k)$ in expected time $O(ns + (r/n)k)$ for $r \leq n/\log^{(s)} n$.*

Taking s as a constant gives us such a $(1/r)$-cutting of expected size $O(r+(r/n)^2 k)$ in expected time $O(n + (r/n)k)$, and taking $s = \log^* n$ gives us a $(1/r)$-cutting of the same expected size (but with a potentially larger r) in expected time $O(n \log^* n + (r/n)k)$ for any $r \leq n$. Since, in our applications involving restrained geometric graphs, k is sublinear in n by an iterated logarithmic factor, we will be taking s to be a constant.

5. Planarization. In this section, we describe how to planarize a connected geometric graph G having n vertices and k edge crossings. We begin by using the method of Theorem 4.3 to construct a $(1/r)$-cutting, C, of the edges of G of expected size $O(r + (r/n)^2 k)$ in expected time $O(n + (r/n)k)$, where $r = n/\log^{(c+1)} n$, for a fixed constant $c \geq 1$. We then do a depth-first search of G, keeping track of the triangles we cross in C as we go, to compute, for each triangle t in C, the set, $C(t)$, of at most (n/r) edges of G that intersect t. This takes $O(|C|n/r)$ time, which has expectation $O(n + (r/n)k)$.

We then apply Goodrich's separator decomposition algorithm [33] to construct an $O(\sqrt{|D|})$-separator decomposition of the graph-theoretic dual, D, to C. Rather than taking this decomposition all the way to the point where we would have subgraphs of D of constant size, however, we stop when subgraphs have size $O(\log^2(n/r))$; hence, we have separators of size $O(\log(n/r))$. We now show how to use these separators to divide our graph into regions for further processing [18, 19, 52]. Since C is a triangulation, D has degree 3; hence, any vertex separator for D of size g also gives us an edge separator for D of size at most $3g$. Moreover, each edge of D corresponds to a triangle edge in C, which in turn crosses at most (n/r) edges of G. For each

separator H in our decomposition, therefore, we can sort the edges of G that cross each boundary of a triangle in the separator in $O((n/r)\log(n/r))$ time. There are $O(|D|/\log^2(n/r))$ nodes at this level of the separator decomposition tree; hence, there are $O(|D|/\log^2(n/r)) \times O(\log(n/r)) = O(|D|/\log(n/r))$ triangles involved. Thus, the total time for all these sorts is $O(|D|(n/r)) = O(n + k(r/n))$, which is $O(n + k)$.

After performing the sorts of edges on the boundaries of triangles in our separators, we can imagine that we have used these boundaries to cut G into $O(|D|/\log(n/r))$ regions (including each triangle in one of our separators) such that the edges of G intersecting each region boundary are given in sorted order. (See Figure 5.1.)

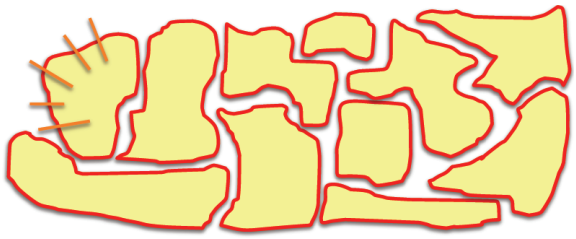

FIG. 5.1. *Illustrating the regions and their boundary edges.*

The total size of each subgraph is $O((n/r)\log^2(n/r))$. Moreover, the boundaries of these regions form a planar subdivision. Thus, we have just subdivided our geometric graph G into $O(|D|/\log(n/r))$ disjoint geometric graphs. In other words, all k edge crossings in G have been isolated into these small subgraphs.

For each subgraph G_i, use Chazelle's algorithm [8] to test whether all the faces of G_i are simple in $O(|G_i|)$ time. If all the faces of G_i are in fact simple, then G_i clearly contains no edge crossings. Thus, we can identify each small subgraph in this partition that contains an intersection in time $O(|C|(n/r) + |G|)$, which has expectation $O(n + k)$.

Clearly, there are at most k such subgraphs that contain edge crossings. We complete our planarization algorithm, therefore, by running the Bentley–Ottmann algorithm [6] for each subgraph of G that is identified as having at least one edge crossing. The time for each such invocation of the Bentley–Ottmann algorithm is $O((n/r)\log^3(n/r) + k'\log(n/r))$, where $k' \geq 1$ is the number of edge crossings found. Summing this over k regions implies that the total time needed to complete the planarization of G is $O(k(n/r)\log^3(n/r))$. Substituting for r, we see that this time is $O(k\log^{(c+1)} n \log^3 \log^{(c+1)} n)$, which is $O(k\log^{(c)} n)$. Therefore, we have the following.

THEOREM 5.1. *Suppose one is given a connected geometric graph G with n vertices and k edge crossings, together with a $(1/r)$-cutting of the edges of G of size $O(r + (r/n)^2 k)$ for $r = n/\log^{(c+1)} n$. Then one can construct a planarization of G (and the trapezoidal decomposition of the arrangement of G's edges) in time $O(n + k\log^{(c)} n)$.*

Combining this result with Theorem 4.3, we get the following corollary.

COROLLARY 5.2. *Given a connected geometric graph G having n vertices and k pairwise edge crossings, one can construct a planarization of G in expected time $O(n + k\log^{(c)} n)$.*

6. Applications. In this section, we provide a number of applications of the above algorithms.

6.1. Separator decompositions and restrained geometric graphs. The algorithms in this section are based on the use of separators. As mentioned above, the separator-decomposition algorithm of Goodrich [33] applies only to planar graphs. We now show how to apply Goodrich's result, together with the tool of planarization, for use with restrained geometric graphs. We do not compute a separator decomposition of restrained geometric graphs directly, however. Instead, we provide a procedure to augment a given restrained geometric graph with dummy vertices in a way that preserves shortest paths, and we then produce a separator decomposition of the resulting augmented graph. In the next subsection, we use this result to find single-source shortest paths and Voronoi diagrams in linear time.

Given a restrained geometric graph G, we planarize it using the algorithm above, creating the planar graph G'. As observed above, G' has total size $O(n)$. Thus, we can use the result of Goodrich [33] to compute a recursive $O(\sqrt{n})$-separator decomposition of G' in $O(n)$ time. Using this separator decomposition of G' and the original graph G, we form an augmentation of G, which we call $\hat{G}$, that preserves shortest paths from G, and we construct an $O(\sqrt{n})$-separator decomposition of $\hat{G}$. We produce this augmentation and separator decomposition by the following transformation. For each vertex v in a separator W of G' at a node w in the separator decomposition tree B, we form a corresponding separator $\hat{W}$ of $\hat{G}$ at a node $\hat{w}$ in $\hat{B}$ as follows:

- If v is also a vertex in G, then we add v to separator $\hat{W}$, provided that v is not already a member of a separator associated with an ancestor of $\hat{w}$.
- If v is an intersection point in G', between edges (a, b) and (c, d) in G, then we add each of a, b, c, and d to separator $\hat{W}$, provided that it is not already a member of a separator associated with an ancestor of $\hat{w}$.
- If no vertex associated with v is added to separator $\hat{W}$, then the balance condition for $\hat{B}$ may be violated. To preserve the balance condition, we create a dummy vertex u in $\hat{G}$, add it to the separator $\hat{W}$, and add an edge of positive weight from u to a vertex in the separator associated with the parent of $\hat{w}$.

It is easy to see that shortest paths in G are preserved in $\hat{G}$ since no dummy vertex is on a simple path between corresponding vertices in G. This gives us the following result.

THEOREM 6.1. *Suppose that we are given an n-vertex geometric graph, G, and its planarization, G', which is of size $O(n)$. We can form a graph $\hat{G}$ which preserves shortest paths in G and construct a recursive $O(\sqrt{n})$-separator decomposition of $\hat{G}$ in $O(n)$ time for $\delta = 2/3$.*

6.2. Single-source shortest paths and Voronoi diagrams. Given an n-vertex bounded-degree graph G and a recursive $O(\sqrt{n})$-separator decomposition for G, Henzinger et al. [37] show that one can compute shortest paths from a single source s in G to all other vertices in G in $O(n)$ time. Applying the shortest-paths-preserving augmentation and the separator decomposition algorithm presented above, then, their algorithm applies to restrained geometric graphs, even ones that do not have bounded degree, by a simple transformation that replaces high-degree vertices with bounded-degree trees of zero-weight edges.

Suppose we are given K distinguished vertices in an n-vertex restrained geometric graph G and we wish to construct the *Voronoi diagram* of G, which is a labeling of each vertex v of G with the name of the distinguished vertex closest to v. As before, by replacing high-degree vertices with bounded-degree trees of zero-weight edges, we can assume without loss of generality that G has constant degree. In this case, we

apply the shortest-paths-preserving augmentation of G and construct its recursive $O(\sqrt{n})$-separator decomposition using the algorithm of the previous subsection. Let B be the recursion tree, and let us label each vertex v in G with the internal node w in B where v is added to the separator or with the leaf w in B corresponding to a set containing v where we stopped the recursion (because the set's size was below our stopping threshold). Given this labeling, we can trace out the subtree B' of B that consists of the union of paths from the root of B to the distinguished vertices in G in $O(n)$ time. Let us now assign each edge in B' to have weight 0, and let us add B' to the augmentation of G to create a larger graph G'. Note that if we add each internal node v in B' to the separator associated with node v in B, then we get a recursive $O(\sqrt{n})$-separator decomposition for G', for each separator in the original decomposition increases by at most one vertex. Thus, we can apply the algorithm of Henzinger et al. [37] to compute the shortest paths in G' from the root of B' to every other vertex in G' in $O(n)$ time. Moreover, since the edges of G' corresponding to edges of B' have weight 0, this shortest path computation will give us the Voronoi diagram for G. Therefore, we have the following.

THEOREM 6.2. *Given a connected n-vertex restrained graph G, together with its planarization, one can compute shortest paths from any vertex s or the Voronoi diagram defined by any set of K vertices in G in $O(n)$ time.*

Incidentally, the above approach also implies a linear-time Voronoi diagram construction algorithm for planar graphs, which was not previously known.

7. Conclusions and future work. We have provided linear-time algorithms for a number of problems on connected restrained geometric graphs, which include real-world road networks. Our results allow for linear-time trapezoidalization, triangulation, and planarization of geometric graphs except for the very narrow range of the number of crossings for which neither our algorithm nor the previous $O(n \log^* n + k)$ algorithm is linear. In addition, our methods imply linear-time algorithms for other problems on such graphs as well. For example, one can use our algorithm to planarize a restrained nonsimple polygon and then construct its convex hull in linear time by computing the convex hull of the outer face of our planarization (e.g., by an algorithm from [35, 41]). There are a number of interesting open problems and future research directions raised by this paper, including the following:

- Can we close the $\log^{(c)} n$ gap on values of k that admit optimal solutions to Chazelle's open problem of computing a trapezoidal decomposition of an n-vertex nonsimple polygon in $O(n + k)$ time, where k is the number of its edge crossings?
- Can we planarize restrained geometric graphs deterministically in linear time? Such a result would allow us to apply separator-based divide-and-conquer techniques for minimum spanning trees [24] to construct them in linear time for this family of graphs. Known linear-time minimum spanning tree algorithms for arbitrary graphs require randomization [39], and known deterministic algorithms for this problem are superlinear [9], although deterministic linear-time algorithms are known for planar graphs and minor-closed graph families [12, 22, 43].

Acknowledgments. We would like to thank Bernard Chazelle for several helpful discussions regarding possible approaches to solving his open problem involving nonsimple polygons. We would also like to thank the anonymous referees for their helpful comments.

REFERENCES

[1] P. K. Agarwal, *Geometric partitioning and its applications*, in Computational Geometry: Papers from the DIMACS Special Year, J. E. Goodman, R. Pollack, and W. Steiger, eds., AMS, Providence, RI, 1991, pp. 1–38.

[2] R. J. Anderson and G. L. Miller, *An optimal algorithm for intersecting line segments in the plane*, Algorithmica, 6 (1991), pp. 859–868.

[3] D. Armon and J. Reif, *A dynamic separator algorithm*, in Proceedings of the 3rd Workshop on Algorithms and Data Structures, Lecture Notes Comput. Sci. 709, Springer-Verlag, New York, Berlin, 1993, pp. 107–118.

[4] F. Aurenhammer, *Voronoi diagrams: A survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405.

[5] F. Aurenhammer and R. Klein, *Voronoi diagrams*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science Publishers/North–Holland, Amsterdam, 2000, pp. 201–290.

[6] J. L. Bentley and T. A. Ottmann, *Algorithms for reporting and counting geometric intersections*, IEEE Trans. Comput., C-28 (1979), pp. 643–647.

[7] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, and M. Yvinec, *Applications of random sampling to on-line algorithms in computational geometry*, Discrete Comput. Geom., 8 (1992), pp. 51–71.

[8] B. Chazelle, *Triangulating a simple polygon in linear time*, Discrete Comput. Geom., 6 (1991), pp. 485–524.

[9] B. Chazelle, *A minimum spanning tree algorithm with inverse-Ackermann type complexity*, J. ACM, 47 (2000), pp. 1028–1047.

[10] B. Chazelle and H. Edelsbrunner, *An optimal algorithm for intersecting line segments in the plane*, J. ACM, 39 (1992), pp. 1–54.

[11] B. Chazelle and L. J. Guibas, *Fractional cascading:* I. *A data structuring technique*, Algorithmica, 1 (1986), pp. 133–162.

[12] D. Cheriton and R. E. Tarjan, *Finding minimum spanning trees*, SIAM J. Comput., 5 (1976), pp. 724–742.

[13] K. L. Clarkson, R. Cole, and R. E. Tarjan, *Erratum: Randomized parallel algorithms for trapezoidal diagrams*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 341–343.

[14] K. L. Clarkson, R. Cole, and R. E. Tarjan, *Randomized parallel algorithms for trapezoidal diagrams*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 117–133.

[15] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry,* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed., MIT Press, Cambridge, MA, 2001.

[17] M. de Berg and O. Schwarzkopf, *Cuttings and applications*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 343–355.

[18] F. Dehne, X. Deng, P. Dymond, A. Fabri, and A. A. Khokhar, *A randomized parallel* 3*d convex hull algorithm for coarse grained multicomputers*, in Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '95), New York, 1995, ACM, New York, pp. 27–33.

[19] X. Deng and B. Zhu, *A randomized algorithm for Voronoi diagram of line segments on coarse grained multiprocessors*, in Parallel Processing Symposium, 1996, Proceedings of the 10th International Parallel Processing Symposium, Honolulu, HI, 1996, IEEE Computer Society Press, Piscataway, NJ, pp. 192–198.

[20] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing*, Prentice–Hall, Upper Saddle River, NJ, 1999.

[21] H. Edelsbrunner, J. W. Jaromczyk, J. W. Penny, and G. Świątek, *Primal canoes: Optimal arrangements of segments*, in Proceedings of the 3rd Canadian Conference on Computational Geometry, Burnaby, BC, 1991, pp. 224–227.

[22] D. Eppstein, *Spanning trees and spanners*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier, New York, 2000, pp. 425–461.

[23] D. Eppstein, *Setting parameters by example*, SIAM J. Comput., 32 (2003), pp. 643–653.

[24] D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer, *Separator based sparsification.* I. *Planarity testing and minimum spanning trees*, J. Comput. Systems Sci., 52 (1996), pp. 3–27.

[25] D. Eppstein and M. T. Goodrich, *Studying (non-planar) road networks through an algorithmic lens*, in Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '08), New York, 2008, ACM, New York,

pp. 1–10.
[26] D. Eppstein, M. T. Goodrich, and D. Strash, *Linear-time algorithms for geometric graphs with sublinearly many crossings*, in Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '09), New York, 2009, SIAM, Philadelphia, pp. 150–159.
[27] D. Eppstein, G. L. Miller, and S.-H. Teng, *A deterministic linear time algorithm for geometric separators and its applications*, in Proceedings of the 9th Annual ACM Symposium on Computational Geometry, San Diego, CA, 1993, ACM, New York, pp. 99–108.
[28] M. Erwig, *The graph Voronoi diagram with applications*, Networks, 36 (2000), pp. 156–163.
[29] J. W. Essam and M. E. Fisher, *Some basic definitions in graph theory*, Rev. Modern Phys., 42 (1970), pp. 271–288.
[30] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.
[31] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, in Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93), Austin, TX, 1993, SIAM, Philadelphia, pp. 222–231.
[32] A. V. Goldberg and C. Harrelson, *Computing the shortest path: A* search meets graph theory*, in Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '05), Vancouver, BC, 2005, SIAM, Philadelphia, pp. 156–165.
[33] M. T. Goodrich, *Planar separators and parallel polygon triangulation*, J. Comput. System Sci., 51 (1995), pp. 374–389.
[34] M. T. Goodrich and R. Tamassia, *Algorithm Design: Foundations, Analysis, and Internet Examples*, John Wiley & Sons, New York, 2002.
[35] R. L. Graham and F. F. Yao, *Finding the convex hull of a simple polygon*, J. Algorithms, 4 (1983), pp. 324–331.
[36] S. Har-Peled, *Constructing planar cuttings in theory and practice*, SIAM J. Comput., 29 (2000), pp. 2016–2039.
[37] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, *Faster shortest-path algorithms for planar graphs*, J. Comput. System Sci., 55 (1997), pp. 3–23.
[38] M. Holzer, F. Schulz, D. Wagner, and T. Willhalm, *Combining speed-up techniques for shortest-path computations*, J. Exp. Algorithmics, 10 (2005), article 2.5.
[39] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, J. ACM, 42 (1995), pp. 321–328.
[40] G. A. Klunder and H. N. Post, *The shortest path problem on large-scale real-road networks*, Networks, 48 (2006), pp. 182–194.
[41] D. T. Lee, *On finding the convex hull of a simple polygon*, Internat. J. Comput. Inform. Sci., 12 (1983), pp. 87–98.
[42] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
[43] M. Mareš, *Two linear time algorithms for MST on minor closed graph classes*, Arch. Math. (Brno), 40 (2004), pp. 315–320.
[44] K. Mehlhorn, *A faster approximation algorithm for the Steiner problem in graphs*, Inform. Process. Lett., 27 (1988), pp. 125–128.
[45] U. Meyer, *Average-case complexity of single-source shortest-paths algorithms: Lower and upper bounds*, J. Algorithms, 48 (2003), pp. 91–134.
[46] G. L. Miller, *Finding small simple cycle separators for 2-connected planar graphs*, J. Comput. System Sci., 32 (1986), pp. 265–279.
[47] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Separators for sphere-packings and nearest neighbor graphs*, J. ACM, 44 (1997), pp. 1–29.
[48] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis, *Geometric separators for finite-element meshes*, SIAM J. Sci. Comput., 19 (1998), pp. 364–386.
[49] B. Mohar and C. Thomassen, *Graphs on Surfaces*, Johns Hopkins University Press, Baltimore, 2001.
[50] J. Pach, *Towards a Theory of Geometric Graphs*, Contemp. Math. 342, AMS, Providence, RI, 2004.
[51] R. Raman, *Recent results on the single-source shortest paths problem*, SIGACT News, 28:2 (1997), pp. 81–87.
[52] E. A. Ramos, *An optimal deterministic algorithm for computing the diameter of a three-dimensional point set*, Discrete Comput. Geom., 26 (2001), pp. 233–244.
[53] P. Sanders and D. Schultes, *Highway hierarchies hasten exact shortest path queries*, in Proceedings of the 17th European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 3669, Springer, New York, 2005, pp. 568–579.

[54] R. Sedgewick and J. S. Vitter, *Shortest paths in Euclidean graphs*, Algorithmica, 1 (1986), pp. 31–48.
[55] D. A. Spielman and S.-H. Teng, *Disk packings and planar separators*, in Proceedings of the 12th Annual ACM Symposium on Computational Geometry, ACM, New York, 1996, pp. 349–358.
[56] M. Thorup, *Undirected single-source shortest paths with positive integer weights in linear time*, J. ACM, 46 (1999), pp. 362–394.
[57] W. T. Trotter, *Planar Graphs*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 9, AMS, Providence, RI, 1993.
[58] F. B. Zhan and C. E. Noon, *Shortest path algorithms: An evaluation using real road networks*, Transportation Sci., 32 (1998), pp. 65–73.