Shortest Paths in Geometric Intersection Graphs

by

Dimitrios Skrepetos

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Norbert Zeh Professor, Faculty of Computer Science, Dalhousie University
Supervisor(s):	Timothy M. Chan Professor, Department of Computer Science, University of Illinois at Urbana-Champaign Adjunct Professor, Cheriton School of Computer Science, University of Waterloo
	Anna Lubiw Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Ian Munro Professor, Cheriton School of Computer Science, University of Waterloo
	Eric Blais Assistant Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Joseph Cheriyan Professor, Department of Combinatorial Optimization, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis describes results from the following papers, which were all coauthored by me and Timothy Chan.

- All-pairs shortest paths in unit-disk graphs in slightly subquadratic time [ISAAC 2016]
- All pairs shortest paths in geometric intersection graphs [WADS 2017]
- Faster approximate diameter and distance oracles in planar graphs [ESA 2017]
- Approximate shortest-path problems and distance oracles in weighted unit-disk graphs [SoCG 2018]

Abstract

This thesis studies shortest paths in geometric intersection graphs, which can model, among others, ad-hoc communication and transportation networks. First, we consider two classical problems in the field of algorithms, namely *Single-Source Shortest Paths* (SSSP) and *All-Pairs Shortest Paths* (APSP). In SSSP we want to compute the shortest paths from one vertex of a graph to all other vertices, while in APSP we aim to find the shortest path between every pair of vertices. Although there is a vast literature for these problems in many graph classes, the case of geometric intersection graphs has been only partially addressed.

In unweighted unit-disk graphs, we show that we can solve SSSP in linear time, after presorting the disk centers with respect to their coordinates. Furthermore, we give the first (slightly) subquadratic-time APSP algorithm by using our new SSSP result, bit tricks, and a shifted-grid-based decomposition technique.

In unweighted, undirected geometric intersection graphs, we present a simple and general technique that reduces APSP to static, offline intersection detection. Consequently, we give fast APSP algorithms for intersection graphs of arbitrary disks, axis-aligned line segments, arbitrary line segments, d-dimensional axis-aligned boxes, and d-dimensional axis-aligned unit hypercubes. We also provide a near-linear-time SSSP algorithm for intersection graphs of axis-aligned line segments by a reduction to dynamic orthogonal point location.

Then, we study two problems that have received considerable attention lately. The first is that of computing the *diameter* of a graph, i.e., the longest shortest-path distance between any two vertices. In the second, we want to preprocess a graph into a data structure, called *distance oracle*, such that the shortest path (or its length) between any two query vertices can be found quickly. Since these problems are often too costly to solve exactly, we study their approximate versions.

Following a long line of research, we employ Voronoi diagrams to compute a $(1+\varepsilon)$-approximation of the diameter of an undirected, non-negatively-weighted planar graph in time near linear in the input size and polynomial in $1/\varepsilon$. The previously best solution had exponential dependency on the latter. Using similar techniques, we can also construct the first $(1+\varepsilon)$-approximate distance oracles with similar preprocessing time and space and only $O(\log(1/\varepsilon)/\varepsilon)$ query time.

In weighted unit-disk graphs, we present the first near-linear-time $(1+\varepsilon)$-approximation algorithm for the diameter and for other related problems, such as the radius and the bichromatic closest pair. To do so, we combine techniques from computational geometry and planar graphs, namely well-separated pair decompositions and shortest-path separators. We also show how to extend our approach to obtain $O(1/\varepsilon)$-query-time $(1+\varepsilon)$-approximate distance oracles with near linear preprocessing time and space. Then, we apply these oracles, along with additional ideas, to build a data structure for the $(1+\varepsilon)$-

approximate All-Pairs Bounded-Leg Shortest Paths (apBLSP) problem in truly subcubic time.

Acknowledgements

First, this thesis would not be a reality without my parents, who strove since my childhood to show me the right path. Second, this thesis would not be a reality without Professor Athanasios Tsakalidis, who introduced me to the world of theoretical computer science. Third, this thesis would not be a reality without my supervisor, Professor Timothy M. Chan, who taught me how research is done. Fourth, this thesis might not have been a reality without the interactions I have had over the years with friends and foes, teachers and professors, video games and books, places I lived in or visited, et cetera. For good or for bad, everyone and everything played a role; is that not what people in chaos theory call *butter y e ect*? But that would be the topic of another thesis.

Dedication

This thesis is dedicated to my parents.

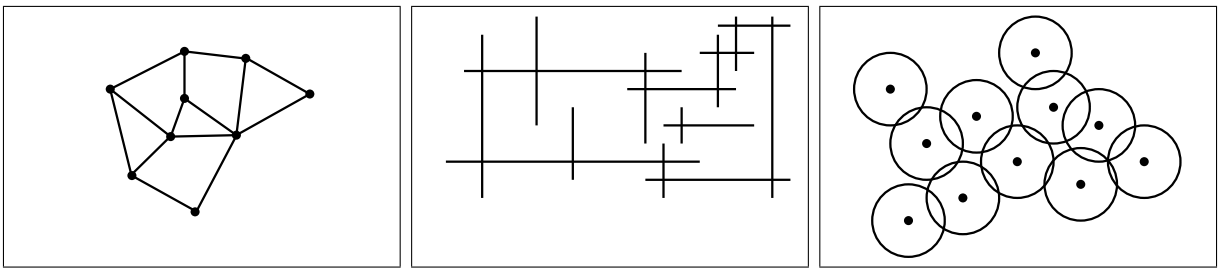
Table of Contents

List of Figures

Chapter 1

Introduction

Shortest-path problems abound in transportation and communication networks, even if we hardly notice them. For example, how can we compute the best route between two cities of a country? Similarly, given a city with roads either perpendicular or parallel to one another, how can we find a path between two addresses with the minimum number of turns? Last, consider a swarm of robots with antennas of unit ranges: how can we upper-bound the time needed to transmit a message from one robot to another? Figure 1.1 depicts an example setting for each question.



(a) (b) (c)

Figure 1.1: A country road network, a city road network, and a swarm of robots (the disks represent the ranges of the antennas).

We can model transportation and communication networks, such as the above, with well-known mathematical structures: graphs. For example, for the country road network in the first question, vertices correspond to cities and edges to roads. Similarly, for the city road network in the second question, vertices correspond to roads and edges to road intersections, and, for the ad-hoc communication network in the third question, vertices correspond to robots and edges to pairs of robots at Euclidean distance at most two. Figure 1.1 illustrates the corresponding graph for each network.

The aim of this thesis is to study shortest-path problems in graphs that can model transportation and communication networks. Specifically, we are interested in the following

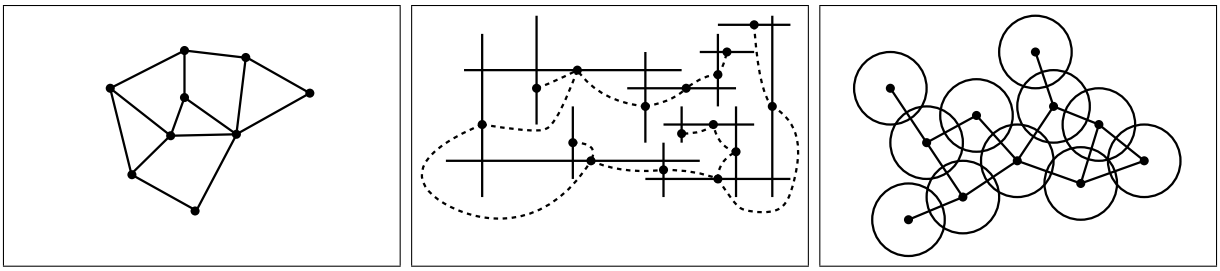


Figure 1.2: The graphs defined for the networks of Figure 1.1.

four problems.

- **Single-source shortest paths** (SSSP) and **all-pairs shortest paths** (APSP). In SSSP we want to compute the shortest paths from a source to all vertices of a graph, while in APSP we want to compute all pairwise shortest paths. Both problems are among the most studied in the field of algorithms, but they have been only partially addressed in the context of communication and transportation networks.

- **Diameter**. In this problem, which has received considerable attention recently due to interesting results in general and planar graphs, we seek to find the longest shortest-path distance of a graph, i.e., its *diameter*. The diameter of a transportation or communication network is important, as it upper-bounds the time needed to travel between any two cities or to exchange a message between any two entities, respectively. Similar problems are computing the farthest neighbor of each vertex of a graph (i.e., the vertex with the longest shortest-path distance from it) and the radius of that graph (i.e., the smallest distance from any vertex to its farthest neighbor).

- **Distance oracles**. Here we intend to preprocess a graph into a data structure, called *distance oracle*, such that we can return the shortest path or its length between any two query vertices. For the rest of Chapter 1, we assume that all queries are of the second type. Those data structures are necessary in modern transportation and communication networks because their gigantic size prohibits us from storing either the network itself (and run SSSP to answer each query) or all pairwise shortest paths (by running APSP in the preprocessing phase).

Next, we survey solutions for the aforementioned problems in general graphs.

1.1 General graphs

Formally, a graph $G = (V, E)$ is defined as a set V of abstract objects, called *vertices*, and a set E of pairs of vertices, called *edges*, which can be either directed or undirected, and

either weighted or unweighted, i.e., have unit weight. Assuming that the vertices of G are numbered $0, \ldots, n-1$, we can represent G either with an $n \times n$ *adjacency matrix* A, where $A[i,j]$ denotes whether $(i,j) \in E$, or with a collection of *adjacency lists*, where the i-th list contains each j such that $(i,j) \in E$. Both representations can naturally be extended to incorporate weights. We assume that the results stated in this section use the second representation. For the rest of Chapter 1, let $\varepsilon > 0$ be a constant, and let n and m be the number of vertices and edges of a graph respectively. We assume that $m \geq n - 1$. In this survey, we focus primarily on graphs of positive edge weights.

Single-source shortest paths. In non-negatively-weighted graphs, the well-known algorithm of Dijkstra [Dij59], implemented with the Fibonacci heaps of Fredman and Tarjan [FT87], requires $O(m + n \log n)$ time. In unweighted graphs, a simple breadth-first search (BFS) takes $O(m+n)$ time.

Polylogarithmic improvements are obtainable in directed graphs with positive integer edge weights in the word RAM model of computation. Let C be the largest edge weight. The problem can be solved in $O(m \log\log C)$ time, either with the deterministic algorithm of Hagerup [Hag00] or with a randomized approach that employs van Emde Boas trees [vEBKZ76] and [vEB77]. The fusion trees [FW93] and the atomic heaps [FW94] of Fredman and Willard imply algorithms of $O(m \log n)$ and $O(m + n \log n / \log\log n)$ expected time respectively. Dial's algorithm [Dia69] takes $O(m + nC)$ time, which was improved to $O(m + n \log C)$ by Ahuja et al. [AMOT90], to $O(m + n\sqrt{\log C})$ expected, by Cherkassky et al. [CGS99], and to $O(m + n\sqrt[4]{\log C})$ by Raman [Ram97]. The priority queues of Thorup [Tho00] can be employed to solve SSSP in $O(m\sqrt{\log n})$ and $O(m \log\log n)$ expected time, the latter of which was derandomized by Hagerup [Hag00]. Han [Han01] presented a deterministic $O(m \log\log n \log\log\log n)$-time algorithm, and Raman [Ram97] developed two deterministic and one randomized algorithms that run in $O(m + n\sqrt[3]{\log n \log\log n})$, $O(m + n\sqrt{\log n \log\log n})$, and $O(m + n\sqrt{\log n})$ time respectively. The best upper bound so far, $O(m + n \log\log \min\{n, C\})$, belongs to Thorup [Tho04b], who has also given a linear-time algorithm for undirected graphs [Tho99].

Graphs with negative edge weights have also been studied. For example, the classical Bellman-Ford algorithm takes $O(mn)$ time. For more such results, see [Gar85, GT89, Gol95, San05, CMSV17].

All-pairs shortest paths. In real-weighted graphs, the classical Floyd–Warshall algorithm can solve APSP in $O(n^3)$ time. The first improvement came by Fredman [Fre76], who gave an $O\left(n^3 \left(\frac{\log\log n}{\log n}\right)^{1/3}\right)$-time algorithm and inspired a number of results that shave off polylogarithmic factors (see [Dob90, Tak92, Han04, Tak04, Tak05, Zwi06, Cha08, Han08, Cha10b, HT12]). Recently, Williams [Wil14], in a breakthrough, provided the first superpolylogarithmic speedup with a randomized algorithm of $\frac{n^3}{2^{\Omega(\sqrt{\log n})}}$ time, which was

later derandomized by Chan and Williams [CW16]. However, obtaining a truly-subcubic-time APSP algorithm, i.e., one that runs in $O(n^3/\log^{1+\varepsilon} n)$ time for some constant $\varepsilon > 0$, still remains a big open problem.

Similarly to SSSP, better results exist for APSP when the edge weights are small positive integers, this time by using matrix multiplication. Specifically, in unweighted, undirected graphs, Seidel [Sei95] and Galil and Margalit [GM97] presented $\tilde{O}(n^\omega)$-time algorithms, where $\omega < 2.373$ [Wil12, Le 14] is the matrix multiplication exponent, and $\tilde{O}(f(n,m))$ denotes $O(f(n,m)\log^{O(1)} n)$. They also showed how to handle the case of positive integer weights upper-bounded by C in $\tilde{O}(Cn^{(\omega+1)/2})$ time, which was improved to $\tilde{O}(Cn^\omega)$ by Shoshan and Zwick [SZ99]. In unweighted, directed graphs, Alon et al. [AGM97] solved the problem in $\tilde{O}(n^{(3+\omega)/2})$ time. For graphs with edge weights upper-bounded by C, their algorithm requires $\tilde{O}(Cn^{(3+\omega)/2})$ time if $C \le n^{3(\omega-2)/(5-\omega)}$ or $\tilde{O}(Cn^{(3+\omega)/2})$ time otherwise. Takaoka [Tak98] improved upon Alon et al. with an $\tilde{O}(Cn^{(6+\omega)/3})$-time algorithm. Finally, Zwick's [Zwi02] algorithm works for edge weights in $\{-n^t, \ldots, 0, \ldots, n^t\}$ and requires $\tilde{O}(n^{2+\alpha(t)})$ time, where $\alpha(t)$ satisfies the equation $\omega(1, \alpha(t), 1) = 1 + 2t$, and $\omega(1, \alpha, 1)$ is the exponent of the multiplication of an $n \times n^\alpha$ matrix by an $n^\alpha \times n$ matrix. The best bound on $\omega(1, \alpha, 1)$, due to Le Gall [LG12], implies that for weights in $\{-1, 0, 1\}$ the problem can be solved in $\tilde{O}(n^{2.5302})$ time.

In sparse graphs, Johnson's algorithm [Joh77] solves APSP in $O(mn + n^2 \log n)$ time, while Hagerup [Hag00] gave an $O(mn + n^2 \log\log n)$-time algorithm in the word RAM. The same result can be obtained by running Thorup's algorithm n times [Tho03] and was extended to the real RAM model of computation by Pettie [Pet04]. Pettie and Ramachandran [PR05] developed a slightly faster algorithm for undirected graphs, requiring $O(mn \log \alpha(m,n))$ time, where α denotes the inverse Ackermann function. Finally, in the word RAM, Chan [Cha10b] showed how to achieve a polylogarithmic improvement over the naive $O(mn)$-time approach of running n BFSs. For more results on SSSP and APSP, see the survey of Zwick [Zwi01].

Diameter. Surprisingly, the only known way of computing the diameter of a graph exactly is to naively run one of the above APSP algorithms and return the largest shortest-path distance found. In fact, it is still open whether the diameter problem is as hard as APSP, although many other related problems, such as computing the radius [AGW15], have been proven to be APSP-hard.

Fortunately, faster algorithms exist in the approximate setting. For the rest of Chapter 1, a c-approximation of the diameter Δ is a value $\hat{\Delta}$ such that $\frac{1}{c}\Delta \le \hat{\Delta} \le \Delta$. Besides a trivial 2-approximation in undirected graphs, which can be obtained with any SSSP algorithm, Aingworth et al. [ACIM99] provided the first such result. Their algorithm works in non-negatively-weighted, directed graphs and computes a $3/2$-approximation in $\tilde{O}(n^2 + m\sqrt{n})$ time. Thus, even in dense graphs the diameter can be approximated in truly subcubic time. Boitmanis et al. [BFLO06] gave an $\tilde{O}(m\sqrt{n})$-time algorithm in unweighted,

undirected graphs that computes an *additive* approximation $\hat{\Delta}$ with $\Delta \leq \hat{\Delta} \leq \Delta + \sqrt{2}\bar{n}$. A similar result can also be obtained with random sampling.

Roditty and Williams [RVW13] developed the first truly-subquadratic-time $(2+\varepsilon)$-approximation algorithm in sparse graphs, with $\tilde{O}(m\sqrt{n})$ expected running time and approximation factor the same as that in [ACIM99]. They also argued that in unweighted, undirected sparse graphs that factor is most likely tight, by proving that unless the Strong Exponential Time Hypothesis (SETH) fails, one cannot obtain a $(3/2-\varepsilon)$-approximation in $O(n^2)$ time. Finally, Chechik et al. [CLR 14] gave a 3/2-approximation algorithm that runs in $\tilde{O}(\min\{m^{3/2}, mn^{2/3}\})$ deterministic time.

Distance oracles. There are two obvious approaches to construct exact distance oracles. In the first, we run an APSP algorithm in the preprocessing phase, store all pairwise shortest paths, and use them to answer each query. In the second, we skip the preprocessing and answer a query by running an SSSP algorithm on the fly.

The approximate setting does not help in directed graphs, as it has been proven that any distance oracle with a finite approximation factor can also answer reachability queries, for which no worst-case efficient data structure is known [AF90, Păt11]. However, the undirected case does admit approximation results, the most prominent of which is by Thorup and Zwick [TZ05]. For the rest of Chapter 1, a *c-approximate distance oracle* of a graph $G = (V, E)$ can return an approximation $\hat{d}(u, v)$ of the u-to-v shortest-path distance $dist_G(u, v)$ with $dist_G(u, v) \leq \hat{d}(u, v) \leq c\,dist_G(u, v)$ for any $u, v \in V$. Thorup and Zwick showed that given an integer $k \geq 1$, a $(2k-1)$-approximate distance oracle of $O(kn^{1+1/k})$ space and $O(k)$ query time can be constructed in $O(kmn^{1/k})$ expected time. That result was later derandomized by Roditty et al. [RTZ05]. Oracles of similar space and approximation factor had been developed previously [ADD 93, Mat96, ABCP98, Coh98, DHZ00], but their query time was much worse, namely $\tilde{O}(kn^{1/k})$.

Using the widely-believed and partially-proven Erdös's girth conjecture, Thorup and Zwick argued that the space/approximation-factor trade-off of their oracle is optimal. Thus, subsequent results on $(2k-1)$-approximate oracles were focused on improving either the query or the preprocessing time. In the former direction, Chechik [Che14] and Wulff-Nilsen [WN13a] developed oracles with $O(1)$ and $O(\log k)$ query time respectively. In the latter direction, Wulff-Nilsen [WN13a], Baswana and Sen [BS06], and Baswana and Kavitha [BK10] showed how to construct oracles of $O(kn^{1+1/k})$ space in near-quadratic time. The oracle of Baswana and Sen works only for the unweighted case, while that of Baswana and Kavitha works only for $k \geq 3$. For more results on oracles, see the survey of Sommer [Som14].

Despite the abundance of shortest-path results in general graphs, they are not efficient enough for modern communication and transportation networks. First, these networks could be dense, e.g., consider a city where each pair of horizontal and vertical roads intersect

each other, as in Figure 1.3. We would need to spend $O(n^2)$ time to solve SSSP, which is too slow for practical purposes. Second, the best solutions for the diameter and the distance oracles problems give 1.5- and 3-approximations respectively, so they are far from useful in transportation networks. For example, according to Google Maps the shortest-path distance between New York and Los Angeles in the road network of USA is 2,777 miles, so with a 3-approximate distance oracle we might need to drive 5,554 extra miles(!).

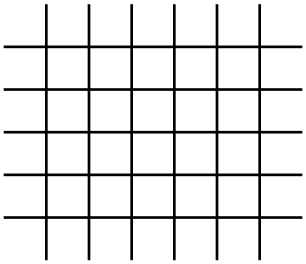


Figure 1.3: A dense city road network that is represented by the intersection graph of axis-aligned line segments.

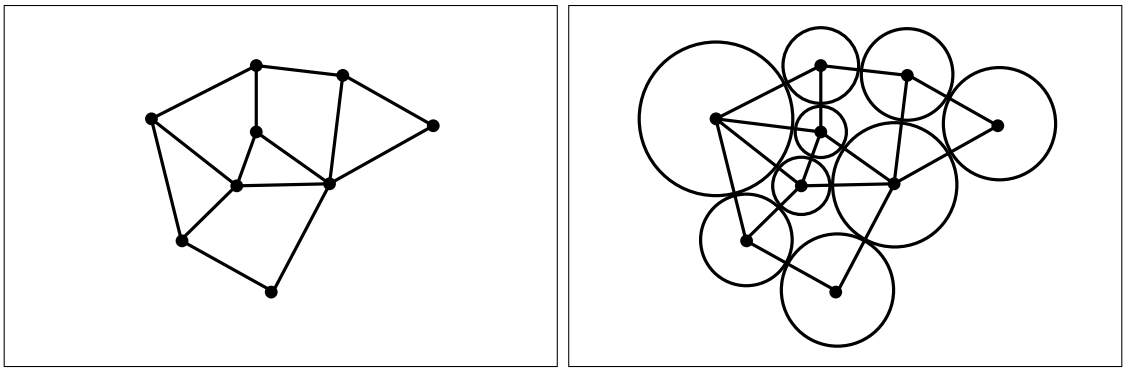
Fortunately, we can achieve much better solutions by noticing that often communication and transportation networks can be represented by *geometric intersection graphs*. These graphs are intersection graphs of geometric objects, where each vertex corresponds to an object and each edge to a pairwise intersection. For example, the roads of the country road network in Figure 1.1(a) intersect only at cities (assuming that there are no bridges), so we can model it with a weighted, undirected *planar graph*,[1] i.e., a graph that can be embedded in the plane without edge crossings. In that graph, vertices correspond to cities, edges to roads, and edge weights to road lengths; see Figure 1.4. Similarly, we can use an unweighted *intersection graph of axis-aligned line segments* for the city road network in Figure 1.1(b), where segments correspond to roads, and a weighted *unit-disk graph* for the ad-hoc communication network in Figure 1.1(c), where unit disks are centered at robots.

Henceforth, we focus on studying shortest-path problems in geometric intersection graphs. Next, we discuss known results for planar graphs in Section 1.2 and for other classes of geometric intersection graphs in Section 1.3.

1.2 Planar graphs

A planar graph is a graph that can be drawn in the plane (i.e., map each vertex to a planar point and each edge to a curve), such that edges intersect only at their endpoints. Euler's formula implies that these graphs are sparse, namely $m \leq 3n - 6$, so they are usually

[1] By the circle packing theorem of Koebe–Andreev–Thurston, any planar graph can be seen as a *coin graph* (i.e., an intersection graph of disks that intersect only on boundaries). There are numerical methods to find an approximate coin-graph representation for a given planar graph [Moh93, CS03], but the disk centers and radii may be irrational numbers [BDEG15].



(a) (b)

Figure 1.4: A planar graph and a coin-graph representation for it.

represented with a collection of adjacency lists. We can test, in linear time, whether a given arbitrary graph is planar with the algorithm of Hopcroft and Tarjan [HT74] or with other more practical methods [BCPDB03, BM04, dFdMR06, dFdM12], and if so, draw it in the plane with straight line segments, as in [DFPP90, Sch90].

A key property of planar graphs, used extensively to provide efficient shortest-path results, is the existence of efficient *separators*: a set of vertices whose removal decomposes the graph into disjoint, induced subgraphs. Lipton and Tarjan [LT79] and Miller [Mil86] showed how to compute in linear time a separator of $O(\sqrt{n})$ vertices that decomposes the graph into two subgraphs of at most $2n/3$ vertices each. Usually, shortest-path algorithms employ separators as a subroutine to build an *r-division* for a specified parameter r, which is defined as a collection of $O(n/r)$ edge-disjoint subgraphs of at most $O(\sqrt{r})$ boundary vertices each (a vertex is a boundary one if it belongs to two or more of these subgraphs). Frederickson [Fre87] showed how to compute such a division in $O(n \log r + (n/\sqrt{r}) \log n)$ time by applying the algorithm of Lipton and Tarjan recursively until graphs of size r are reached. Goodrich [Goo95] improved the time to linear, while subsequent papers tried to improve other quality parameters of the division [Sub95, KS98, FR06, Cab06, CR10, INSWN11, KMS13]. In another direction, Thorup [Tho04a] provided a linear-time algorithm that computes a *shortest-path separator*, i.e., a separator made of $O(1)$ shortest paths that decompose a graph into three subgraphs of at most $n/2$ vertices each. This type of separators has been employed to develop numerous $(1+\varepsilon)$-approximate shortest-path results [Tho04a, Kle02, KKS11, KST13, GX15, WY16].

Single-source shortest paths. In unweighted, directed planar graphs, SSSP can be solved in linear time with BFS, while in the presence of non-negative weights, Dijkstra's algorithm takes $O(n \log n)$ time. For the weighted case, Frederickson [Fre87] employed a recursive scheme of $O(\log n)$ levels, namely the r-division discussed above, to improve the time to $O(n\sqrt{\log n})$. Later, Henzinger et al. [HKRS97] obtained an optimal linear-time solution by using only $O(\log n)$ recursion levels and a complicated version of Dijkstra's

algorithm. The version of SSSP with negative weights has also been studied, where the best result is due to Mozes and Wulff-Nilsen [MWN10] and takes $O(n \log^2 n \log\log n)$ time.

All-pairs shortest paths and diameter. Since Frederickson [Fre87] computed the diameter (by solving APSP) in non-negatively-weighted, directed planar graphs in $O(n^2)$ time, a natural question arose as to whether there exists a subquadratic-time algorithm for the problem. Eppstein [Epp99] gave a partial answer for the unweighted case by proving that if the diameter is constant, it can be found in linear time. Chan [Cha12] and Wulff-Nilsen [WN10] presented two slightly-subquadratic-time solutions (for arbitrary diameter), both requiring $O(n^2 \frac{\log\log n}{\log n})$ time in unweighted graphs. Wulff-Nilsen's algorithm also works for the weighted case but in $O(n^2 \frac{(\log\log n)^4}{\log n})$ time. However, a truly-subquadratic-time algorithm eluded researchers for many years, thus leading them to consider approximation algorithms.

A trivial 2-approximation can be obtained with the linear-time SSSP algorithm of Henzinger et al. [HKRS97], while Berman and Kasiviswanathan [BK07] showed how to compute a 3/2-approximation in $\tilde{O}(n^{3/2})$ time. Weimann and Yuster [WY16], in a breakthrough, presented the first $(1+\varepsilon)$-approximation algorithm, requiring $O(n(1/\varepsilon)^4 \log^4 n \cdot 2^{O(1/\varepsilon)})$ time. Nevertheless, their solution does not settle the problem because of the *exponential* dependency on $1/\varepsilon$ and of the multiple (four) $\log n$ factors.

Unexpectedly, the next result came in the context of exact algorithms. In 2017, Cabello [Cab17a] (full paper in [Cab17b]) made headway, by giving the first *exact* truly-subquadratic-time algorithm, running in $\tilde{O}(n^{11/6})$ expected time. Interestingly, Cabello employed a seemingly alien concept to planar graphs, *Voronoi diagrams*, which originates from computational geometry. Later, Gawrychowski et al. [GKM+18], again using Voronoi diagrams, derandomized Cabello's algorithm and improved its running time to $\tilde{O}(n^{5/3})$.

Distance oracles. In real-weighted, directed planar graphs, Djidjev [Dji96] and Arikati et al. [ACC+96] presented the first exact oracles. That of Djidjev has $\tilde{O}(n/\sqrt{S})$ query time for space $S = \Omega(n^{4/3}), n^{3/2})$, which was subsequently extended to the whole range $S = \Omega(n), n^2)$ [CX00, FR06, Nus11, Cab12, MS12]. Recently, Cohen–Addad et al. [CDW17] and Gawrychowski et al. [GMWWN18] adapted the Voronoi-diagram-based technique of Cabello [Cab17b] to provide the first oracles with truly subquadratic space and polylogarithmic query time. They also obtained better time-space trade-offs: Gawrychowski et al.'s data structure requires $\tilde{O}(n^{1.5}/S)$ query time for $S = \Omega(n), n^{3/2})$ and $O(\log n)$ time for $S = n^{3/2}$.

If we are willing to settle with an approximation, much more efficient results are known. Specifically, in non-negatively-weighted, undirected planar graphs, Thorup [Tho04a] employed shortest-path separators to construct in $O((1/\varepsilon)^2 n \log^3 n)$ time an oracle with

$O(n \log n)$ space and $O(1)$ query time (he also considered the directed case). Thorup's result was later simplified by Klein [Kle02]. Kawarabayashi et al. [KKS11] described an alternative oracle with linear space but $O(\log^2 n)$ query time, and Kawarabayashi et al. [KST13] improved the dependency on $1/\varepsilon$ of the space-query-time product from $1/\varepsilon^2$ to $1/\varepsilon$. Finally, Gu and Xu [GX15] adapted the framework of the approximate diameter algorithm of Weimann and Yuster [WY16] to obtain the first $(1+\varepsilon)$-approximate distance oracle of $O(1)$ query time in the Word RAM. The oracle of Gu and Xu preprocessing time and space are $O(n \log n (1/\varepsilon)^2 \log^3 n \cdot 2^{O(1/\varepsilon)})$ and $O(n \log n (1/\varepsilon) \log n \cdot 2^{O(1/\varepsilon)})$ respectively.

1.3 Geometric intersection graphs

A geometric intersection graph is the intersection graph of a set of geometric objects, i.e., its vertices correspond to objects and edges to pairwise intersections. Some common classes of these graphs are intersection graphs of:

- intervals in the real line, called *interval graphs*,
- unit disks, called *unit-disk graphs*,
- arbitrary disks, called *disk graphs*,
- disks that intersect only on their boundaries, called *coin graphs*,
- curves, called *string graphs*,
- axis-aligned line segments,
- arbitrary line segments,
- d-dimensional axis-aligned boxes, and
- d-dimensional axis-aligned unit hypercubes.

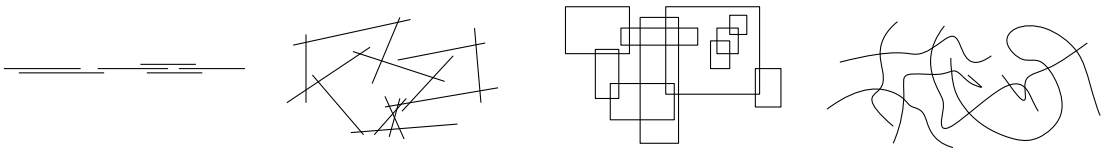


Figure 1.5: Intervals, arbitrary line segments, axis-aligned rectangles, and strings.

Throughout the thesis, we assume that the dimension d is a constant. Although some families of geometric intersection graphs, such as interval graphs [BL76] and arbitrary coin

graphs (i.e., planar graphs), can be *recognized* in polynomial time (i.e., given a graph, determine whether it belongs to a certain family) most cannot. Specifically, the recognition problem has been proved to be NP-hard for unit-disk graphs[2] and for coin graphs of unit disks [BK98], for disk graphs and for coin graphs of constant maximum-to-minimum radii ratio [BK95], for arbitrary disk graphs [HK01], and for intersection graphs of arbitrary segments [KM89]. Moreover, recognizing a string graph is NP-complete [SSŠ03], which is also true for other families of graphs, such as intersection graphs of axis-aligned segments [KM94], of d-dimensional axis-aligned boxes, where $d \geq 2$ [Kra94, Yan82, Coz82], and of d-dimensional axis-aligned unit-hypercubes, where d is 2 or 3 [Yan82].

Embedding geometric intersection graphs (i.e., given a graph G, create a set S drawn from a given family of objects, such that the geometric intersection graph of S is isomorphic to G) is difficult as well. For example, McDiarmid and Müller [MM13] proved that to draw a disk graph on an integer grid, such that each radius is an integer, $2^{2^{pnq}}$ bits are necessary. Their result also holds for unit-disk graphs and intersection graphs of arbitrary segments. Kratochvíl and Matoušek [KM91] showed that there are string graphs that require an exponential number of intersections between their curves. We assume henceforth that geometric graphs are represented implicitly by the set of their corresponding objects (e.g., a unit-disk graph is represented by the set of the coordinates of its disks's centers).

1.3.1 Unit-disk graphs

Since unit-disk graphs can model ad-hoc communication networks, such as that in Figure 1.1(a), and admit many efficient shortest-path algorithms, we make a separate survey for them here.

Single-source shortest paths. Roditty and Segal [RS11] used the ideas of Chan and Efrat [CE01] to reduce the unweighted version of SSSP to dynamic nearest-neighbor searching. Thus, by employing the data structure of Chan [Cha10a], they obtained an $O(n \log^6 n)$-time algorithm. They also reduced the weighted version of SSSP to halfspace range searching in the three-dimensional space [AE99] and solved the problem in $O(n^{4+\varepsilon})$ time.

Cabello and Jejčič [CJ15] improved both results. For the unweighted case, they employed Delaunay triangulation and static additively weighted Voronoi diagrams to provide an $O(n \log n)$-time algorithm. For the weighted case, dynamic Voronoi diagrams are needed, so employing the data structure of Kaplan et al. [KMR+17] yields a solution of nearly $O(n \log^{12} n)$ time.

Diameter and distance oracles. Gao and Zhang [GZ05] extended *well-separated pair decompositions* (WSPD), a well-known technique for proximity problems in computational

[2] Recognizing unit-disk graphs is actually hard for the existential theory of the reals [KM12].

geometry [CK95], from the Euclidean to the weighted unit-disk graph metric. Then, they employed such a WSPD to compute a $(1+\varepsilon)$-approximation of the diameter and to build an $O(1/\varepsilon)$-query-time $(1+\varepsilon)$-approximate distance oracle of near-linear space in $O\left((1+\varepsilon)\varepsilon^{-3}n^{3/2}\sqrt{\log n}\right)$ time.

1.3.2 Other geometric intersection graphs

Single-source shortest paths. Chan and Efrat's techniques [CE01] imply that SSSP in unweighted, undirected geometric intersection graphs can be reduced to *dynamic* data structuring problems. For example, in disk graphs, dynamic additively weighted nearest-neighbor searching data structures are required, and the problem can be solved in nearly $O(n\log^{12} n)$ time [KMR+17]. Moreover, in unweighted, directed disk graphs, Kaplan et al. [KMRS15] showed that SSSP can be solved in $O(n\log n)$ time, after $O(n\log^6 n)$-time preprocessing.

All-pairs shortest paths. There are two general methods for solving APSP in unweighted, undirected geometric intersection graphs. The first runs an SSSP algorithm n times as explained above, thus solving the problem in disk graphs in nearly $O(n^2\log^{12} n)$ time [KMR+17]. The second uses *biclique covers* [FM95, AAAS94], which are related to static, offline intersection searching data structures (e.g., as noted in [Cha10b]), to sparsify the intersection graph and solve APSP therein. For example, sparsifying an intersection graph of d-dimensional boxes (respectively a disk graph) with that method would produce $O(n\log^d n)$ (respectively $O(n^{3/2+\varepsilon})$ [APS93]) edges, leading thus to an $O(n^2\log^d n)$-time (respectively $O(n^{5/2+\varepsilon})$-time) algorithm.

Diameter and distance oracles. Few results are known for diameter and distance oracles in geometric intersection graphs except for some special cases, such as interval graphs, where we can find the diameter [Ola90] and build an $O(1)$-query-time distance oracle [Coh98, ST99] in linear time.

1.4 New results

Even though geometric intersection graphs do admit much better algorithms for shortest-path problems than general graphs do, many issues remain. The main issues we study in this thesis are the following.

- In unweighted unit-disk graphs, APSP can be solved in $O(n^2\log n)$ time [CJ15]. However, in planar graphs, which can be viewed as intersection graphs of disks that only touch on boundaries, the problem can be solved in slightly subquadratic time [Cha12].

- In unweighted, undirected geometric intersection graphs, the general methods that reduce SSSP and APSP to geometric problems require solving either *dynamic* intersection detection [CE01] or static, offline intersection *searching*, both of which are harder than static, offline intersection detection.

- In planar graphs, we can $(1+\varepsilon)$-approximate the diameter [WY16] and build $O(1/\varepsilon)$-query-time $(1+\varepsilon)$-approximate distance oracles in near-linear time [GX15], but the dependency on $1/\varepsilon$ is exponential, and there are multiple (four) logarithmic factors.

- In weighted unit-disk graphs, it takes nearly $O(n^{3/2})$ time [GZ05] to compute a $(1+\varepsilon)$-approximation of the diameter and build $O(1/\varepsilon)$-query-time $(1+\varepsilon)$-approximate distance oracles. However, in planar graphs, only near-linear time is needed for both problems.

1.4.1 Single-source and all-pairs shortest paths

Unweighted unit-disk graphs. In Chapter 3, we show that we can preprocess an unweighted unit-disk graph of n vertices in $O(n \log n)$ time, such that the shortest-path tree from any source can be found in linear time. Consequently, we can solve APSP in quadratic time, improving upon Cabello and Jejčič [CJ15]. Our algorithm uses a grid-based approach and exploits a linear-time Graham-scan-like procedure [PS85] for computing upper envelopes of unit disks that are presorted with respect to the coordinates of their centers.

New Result 1. (SSSP and APSP in unweighted unit-disk graphs) *We can preprocess an unweighted unit-disk graph of n vertices in $O(n \log n)$ time, such that we can solve SSSP for any given source in linear time. Consequently, we can solve APSP in quadratic time.*

Then, we use our SSSP algorithm, bit tricks, and a simple *shifted grid* strategy [HM85] to provide a slightly-subquadratic-time algorithm that computes implicitly the shortest paths of all pairs of vertices and the diameter. Using shifted grids is standard in geometric approximation algorithms, but here we adapt that technique in a new and interesting way to obtain an *exact* solution.

New Result 2. (APSP in slightly subquadratic time in unweighted unit-disk graphs) *We can compute an implicit representation of the shortest paths of all pairs of vertices and the diameter of an unweighted unit-disk graph of n vertices in $O(n^2 \frac{\log\log n}{\log n})$ time.*

Unweighted, undirected geometric intersection graphs. In Chapter 4, we provide a simple and general APSP algorithm for unweighted, undirected geometric intersection graphs by a reduction to static, offline intersection *detection*: given a query object, decide whether there is an input object that intersects it (and report one if the answer is yes). Our

solution uses simpler data structures than the other two general methods, which reduce the problem to dynamic intersection detection and to static, offline, intersection searching respectively.

New Result 3. (APSP in unweighted, undirected geometric intersection graphs) *We can solve APSP in an intersection graph of n disks in $O(n^2 \log n)$ time, axis-aligned line segments in $O(n^2 \log\log n)$ time, arbitrary line segments in $O(n^{7/3} \log^{1/3} n)$ time, d-dimensional axis-aligned boxes in $O(n^2 \log^{d-1.5} n)$ time for $d \geq 2$, d-dimensional axis-aligned unit hypercubes in $O(n^2 \log\log n)$ time for $d = 3$ and $O(n^2 \log^{d-3} n)$ for $d \geq 4$, and fat triangles of roughly equal size in $O(n^2 \log^4 n)$ time.*

We can also solve SSSP in unweighted intersection graphs of n axis-aligned line segments in $O(n \log n)$ time by a reduction to decremental orthogonal point location. Our approach is similar to that of Chan and Efrat [CE01], but simpler because we study the unweighted version of the problem.

New Result 4. (SSSP in unweighted intersection graphs of axis-aligned line segments) *We can solve SSSP in an unweighted intersection graph of n axis-aligned line segments in $O(n \log n)$ time.*

1.4.2 Diameter and distance oracles

Weighted, undirected planar graphs. In Chapter 5, we show how to compute a $(1-\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph of n vertices in $O(n \log n \log\log n \cdot (1/\varepsilon)^{5}(1/\varepsilon))$ time. Hence, we improve upon the $O(n \cdot (1/\varepsilon)^4 \log^4 n \cdot 2^{O(1/\varepsilon)})$-time algorithm of Weimann and Yuster [WY16] in two regards.

First, we eliminate the exponential dependency on $1/\varepsilon$, by adapting and specializing Cabello's recent abstract-Voronoi-diagram-based technique [Cab17a] for approximation purposes. Compared with his algorithm, which had to deal with the general case of site weights being real numbers, our version of Voronoi diagrams is much simplified because in the approximate setting we can map the site weights to small integers. Second, we shave off two logarithmic factors by choosing a better sequence of error parameters in the recursion and by employing the multiple shortest paths data structure of Klein [Kle05].

New Result 5. (Approximate diameter in weighted, undirected planar graphs) *We can compute a $(1-\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph of n vertices in $O(n \log n \log\log n \cdot (1/\varepsilon)^{5}(1/\varepsilon))$ time.*

In the Word RAM, we use similar techniques to improve upon the results of [Tho04a, Kle02, KKS11, KST13] by presenting the first $(1+\varepsilon)$-approximate distance oracle with $o(1/\varepsilon)$ query time and $\tilde{O}((1/\varepsilon)^{O(1)} n)$ preprocessing time and space. The only previous

oracle with $o(1/\varepsilon)$ query time, given by Gu and Xu [GX15], had exponential dependency on $1/\varepsilon$ in the latter.

New Result 6. (Approximate distance oracles in weighted, undirected planar graphs) *We can construct a $(1+\varepsilon)$-approximate distance oracle for a non-negatively-weighted, undirected planar graph of n vertices with $O(n \log^2 n \log\log n)$ $(1/\varepsilon)^{O(1)}$ preprocessing time, $O((1/\varepsilon) n \log^2 n)$ space, and $O(\log\log(1/\varepsilon))$ query time in the Word RAM.*

Weighted unit-disk graphs. In Chapter 6, we show how to obtain the first near-linear-time $(1+\varepsilon)$-approximation algorithm for the diameter of a weighted unit-disk graph of n vertices. Namely, our algorithm runs in $O(n \log^2 n)$ time, for any constant $\varepsilon > 0$, and considerably improves over the near-$O(n^{3/2})$-time algorithm of Gao and Zhang [GZ05]. Using similar ideas, we develop a $(1+\varepsilon)$-approximate *distance oracle* of $O(1/\varepsilon)$ query time with a similar improvement in the preprocessing time, specifically from near $O(n^{3/2})$ to $O(n \log^3 n)$.

The above problems are addressed in planar graphs with divide-and-conquer methods based on the concept of *shortest-path* separator: a set of $O(1/\varepsilon)$ shortest paths with common root, such that the removal of their vertices decomposes the graph into $O(1/\varepsilon)$ disjoint, induced subgraphs whose sizes are at most a constant fraction of the size of the original graph. Even though such separators are not directly applicable to unit-disk graphs (because a path therein may "cross" a separator over an edge), we show that we can still use them. Specifically, we first find a planar $O(1/\varepsilon)$-*spanner* H of the given unit-disk graph G (i.e., H is a planar subgraph of G such that $dist_H(s,t) \le O(1/\varepsilon)\, dist_G(s,t)$ for any two vertices s, t of G) and then compute shortest-path separators therein. Although the spanner has $O(1/\varepsilon)$ approximation factor, we show that we can still obtain $(1+\varepsilon)$-approximate results by using the geometrical properties of unit-disk graphs.

New Result 7. (Approximate diameter in weighted unit-disk graphs) *Given a set S of n planar points, we can compute in $(1/\varepsilon)^{O(1)} n \log^2 n$ $(1/\varepsilon)^{O(1)}$ time a $(1+\varepsilon)$-approximation of the diameter of its weighted unit-disk graph.*

New Result 8. (Approximate distance oracles in weighted unit-disk graphs) *Given a set S of n planar points, we can construct a $(1+\varepsilon)$-approximate distance oracle for its weighted unit-disk graph with $(1/\varepsilon)^{O(1)} n \log^3 n$ $(1/\varepsilon)^{O(1)} n \log(1/\varepsilon)$ preprocessing time, $O((1/\varepsilon) n \log n)$ space, and $O(1/\varepsilon)$ query time.*

As a further application, we employ our new distance oracle, along with additional ideas, to solve the $(1+\varepsilon)$-approximate *all-pairs bounded-leg shortest paths* (apBLSP) problem. Given a set S of n planar points, we define $G_{\le L}$ to be the subgraph of the complete Euclidean graph of S that contains only edges of weight at most L. Then, we want to preprocess S, such that given two points $s, t \in S$ and any positive number L, we can quickly compute a $(1+\varepsilon)$-approximation of length of the s-to-t shortest path in $G_{\le L}$ (i.e., the shortest path under the restriction that each leg of the trip has length bounded by

L). To see the connection of apBLSP with the earlier problems, note that for each fixed L, $G_{\cap L}$ is a weighted unit-disk graph, after rescaling the radii. One important difference however, is that L is not fixed in apBLSP, so we want to be able to answer queries for any of the $\binom{n}{2}$ combinatorially different L's.

Bose et al. [BMN 04] introduced the problem and described a method with $O(n^5)$ preprocessing time, $O(n^2 \log n)$ space, and $O(\log n)$ query time. Roditty and Segal [RS11] improved the preprocessing time to roughly $O(n^3)$ and the query time to $O(\log \log n)$. They also gave a data structure for the variation of the problem in general weighted, directed graphs with $\tilde{O}(n^{2.5})$ space and $\tilde{O}(n^4)$ preprocessing time. Duan and Pettie [DP08] improved the space and the preprocessing time of Roditty and Segal's result in general graphs to $\tilde{O}(n^2)$ and $\tilde{O}(n^3)$ respectively. In a recent independent work that appeared after the conference version of this paper, Duan and Ren [DR18] presented the first data structure for the problem in general graphs with subcubic preprocessing time, namely $\tilde{O}(n^{2.6865})$ (the space remains near quadratic).

We apply our $(1+\varepsilon)$-approximate distance oracle for weighted unit-disk graphs, along with additional new ideas, in Section 6.2 to obtain the first data structure for $(1+\varepsilon)$-approximate apBLSP in the Euclidean metric that breaks the cubic preprocessing barrier given by Roditty and Segal: namely, we obtain nearly $\tilde{O}(n^{2.667})$ preprocessing time, while the space and query time remain $\tilde{O}(n^2)$ and $O(\log \log n)$ respectively as in [RS11]. With fast matrix multiplication, we can further reduce the preprocessing time to $O(n^{2.579})$, assuming a polynomial bound on the *spread*, i.e., the ratio of the maximum to the minimum Euclidean distance over all pairs of points in V. We define the spread of the point set to be the ratio of the maximum-to-minimum pairwise Euclidean distance.

New Result 9. (Approximate all-pairs bounded-leg shortest paths) *Given a set S of n planar points of spread W, we can construct a data structure for the $(1+\varepsilon)$-approximate apBLSP problem with $O\left((1/\varepsilon)^{6(1/\varepsilon)^3} \log^5(nW)\right)$ preprocessing time, $O((1/\varepsilon)n^2 \log n)$ space, and $O(\log \log n + \log(1/\varepsilon))$ query time.*

Finally, we conclude with some open problems on shortest paths in geometric intersection graphs in Chapter 7.

Chapter 2

Preliminaries

2.1 Model of computation

Throughout the thesis, except for Section 5.3, we use the standard (real) RAM model of computation. Specifically, we have random access to an array of words, each storing one of the following: a real number, a $\Theta(\log n)$-bit integer (where n is the input size), or a pointer to another word. Moreover, we can perform any standard arithmetic operation, such as addition, subtraction, multiplication, division, and comparison, that involves a constant number of words in constant time. In Chapter 6, we assume that we can compute square roots of real numbers exactly in constant time. In Chapter 3, we assume that we can compute the floor of an input real number in constant time.

In Chapters 3 and 4, we assume that we can also support custom operations in $(\log n)$-bit words in $O(1)$ time, where $0 < \epsilon < 1$ is a constant, after $O(n)$ preprocessing time. To do so, we perform the desired operation to each of the $2^{\log^{\epsilon} n} = n^{\epsilon}$ possible inputs and store the results in a lookup table (assuming that each operation can naively be performed in $\log^{O(1)} n$ time).

In Section 5.3, we work in the Word RAM model of computation, where the input values are assumed to be w-bit integers ($w \geq \log n$). We assume that standard arithmetic and bit-wise logical operations on w-bit integers take constant time.

2.2 Graphs and shortest-path problems

A graph $G = (V, E)$ consists of a set V of *vertices* and a set E of pair of vertices, called *edges*, which can be either *directed* or *undirected* and either *weighted* or *unweighted* (i.e, have unit weight). Assuming that the vertices of a graph $G = (V, E)$ are numbered $0, \ldots, n-1$, we can represent G either with an $n \times n$ *adjacency matrix* A, where each $A[i, j]$ denotes whether $(i, j) \in E$, or with a collection of *adjacency lists*, where the i-th

list contains each j with $(i,j) \in E$. Both representations can naturally be extended to incorporate weights.

Given a graph $G = (V,E)$ and two vertices $s,t \in S$, an s-to-t path in G is a sequence of vertices $p_0 p_1 \ldots p_\ell$, with $p_0 = s$, $p_\ell = t$, and $(p_i,p_{i-1}) \in E$ for each p_i,p_{i-1}. The length of is defined as $\sum_i w(p_i,p_{i-1})$, where $w(p_i,p_{i-1})$ is the weight of the edge (p_i,p_{i-1}). A shortest path $_G(s,t)$ of s and t in G is defined as an s-to-t path of minimum length, which we denote by $dist_G(s,t)$. Let $pred_G(s,t)$ be t's predecessor on such a path. We define the shortest-path tree $T(s)$ of $s \in V$ to be a spanning tree of G rooted at s, such that the s-to-t shortest-path distance in $T(s)$ corresponds to $dist_G(s,t)$ for each $t \in V$.

Below, we give definitions of some standard shortest-path problems in a graph $G = (V,E)$.

- *Single-source shortest paths (SSSP)*: Given a source $s \in S$, compute $dist_G(s,t)$ and $pred_G(s,t)$ for each $t \in V$.
- *All-pairs shortest paths (APSP)*: Compute $dist_G(s,t)$ and $pred_G(s,t)$ for each $s,t \in V$.
- *Diameter*: Compute $\max_{s,t \in V} dist_G(s,t)$.
- *Wiener index*: Compute $\sum_{s,t \in V} dist_G(s,t)$.
- *Center and radius*: Compute $c = \arg\min_{s \in V} \max_{t \in V} dist_G(s,t)$ and $\min_{t \in V} dist_G(c,t)$ respectively.
- *Farthest neighbors and eccentricities*: Compute $f = \arg\max_{t \in V} dist_G(s,t)$ and $dist_G(s,f)$ respectively for each $s \in V$.
- *Bichromatic closest pair*: Given subsets $A,B \subseteq V$, compute $\min_{a \in A,b \in B} dist_G(a,b)$.
- *Distance oracle*: Preprocess G such that given $s,t \in V$, we can find $dist_G(s,t)$ and $pred_G(s,t)$.
- *All-pairs bounded-leg shortest paths (apBLSP)*: Preprocess G such that given $s,t \in V$ and a number L, we can find $dist_{G \le L}(s,t)$, where $G_{\le L}$ is the subgraph of G that contains only the edges of weight at most L.

2.3 Breadth-first search

The classical breadth-first search (BFS) is a simple algorithm that, among other problems, can also solve SSSP in unweighted graphs. Given a graph $G = (V,E)$ of n vertices and m edges and a source $s \in V$, BFS computes the s-to-t shortest-path distance $dist_G(s,t)$ of each $t \in V$ from s and its predecessor $pred_G(s,t)$ on a s-to-t shortest path. Specifically, BFS repeats the same procedure in n 1 steps, where in the beginning of each step ` it is

assumed that all vertices at distance at most $\ell - 1$ from s have been discovered. We call the rest *undiscovered* vertices and those at distance exactly $\ell - 1$ from s *frontier* vertices. Then, we visit every edge (z, t) incident to a frontier vertex z, and if t is undiscovered, we set its distance from s to ℓ and its predecessor to z. In the end of step ℓ, the frontier vertices are replaced by those at distance ℓ from s, and step $\ell - 1$ begins. See Algorithm 2.1 for the pseudocode. We use a modification of BFS in Chapters 3 and 4 to solve SSSP and APSP in unweighted unit-disk graphs and unweighted, undirected geometric intersection graphs respectively.

Algorithm 2.1: BFS(G, s), where $G = (V, E)$

```
1  dist_G(s, s) ← 0
2  dist_G(s, t) ← ∞, ∀ t ∈ V − {s}
3  pred_G(s, t) ← NULL, ∀ t ∈ V
4  F ← {s}
5  for ℓ ← 1 to n − 1 do
6      F' ← ∅
7      for each z ∈ F do
8          for each edge (z, t) ∈ E do
9              if dist_G(t, s) = ∞ then
10                 dist_G(t, s) ← ℓ
11                 pred_G(t, s) ← z
12                 F' ← F' ∪ {t}
13     F ← F'
14 return dist_G(·, s) and pred_G(·, s)
```

Theorem 2.1. (Breadth-first search) *Given an unweighted graph $G = (V, E)$ of n vertices and m edges, BFS can solve SSSP in $O(m + n)$ time.*

Proof. We prove that BFS correctly solves SSSP with the following inductive hypothesis: at the end of each step ℓ all vertices at distance at most ℓ from s have been discovered. For the base case, $\ell = 0$, our claim is trivially true. From the hypothesis, at the end of each step $\ell - 1$, all vertices at distance at most $\ell - 1$ from s have been discovered. A vertex at distance ℓ from s must share an edge with a vertex at distance $\ell - 1$, but by the end of step ℓ we have properly processed all such edges. Each vertex is a frontier in exactly one step, so each edge is processed twice. Thus the total time is $O(m + n)$. □

2.4 Graham's scan for pseudoline arrangements

Graham's scan [Gra72, dBCvKO08] is a well-known algorithm in computational geometry for computing the convex hull of a planar point set. If we have presorted the points with

respect to their x-coordinates, Graham's scan takes linear time. By duality [dBCvKO08], Graham's scan can also build the upper envelope of a line arrangement whose lines are presorted with respect to their slopes in linear time. Generally, the *upper envelope* of a set of curves consists of their portions that are visible from the point $p0; 8q$ if the curves are viewed as opaque objects. Here, we describe Graham's scan in a more general way, to compute the upper envelope of a *pseudoline* family (see Figure 2.1): a set of curves, such that any two of them cross each other at most once, and any vertical line intersects every curve exactly once.

The pseudocode of the more general version of Graham's scan is given in Algorithm 2.2. We invoke the following theorem in Chapter 3 as a subroutine for computing shortest-path trees in unweighted unit-disk graphs.

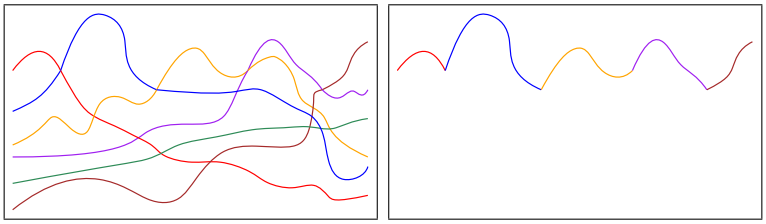


Figure 2.1: A pseudoline arrangement and its upper envelope.

Theorem 2.2. (Graham's scan for pseudoline arrangements) *Given n pseudolines and their order at x 8, we can compute the upper envelope of their arrangement in $O(npt)$ time, assuming that we can compute the intersection point, if any, of two pseudolines in $O(t)$ time.*

Proof Sketch. The correctness of the algorithm can be proven with a simple induction. For the running time, notice that in each iteration i of the main loop of the algorithm there are three cases: we (a) ignore the i-th pseudoline (Line 5), (b) push it in the stack (Lines 8 and 14), or (c) pop the stack's top element (Line 17). The number of iterations of the inner loop is bounded by the number of pops, which in turn can be bounded by the linear number of pushes. Also, in every step of the inner loop, we spend $O(t)$ time to find intersections of pseudolines (Lines 11 and 12). □

2.5 Planar separators and decomposition trees

A *separator* of a planar graph G $(V; E)$ is a subset V^1 of V, such that the removal of its vertices decomposes G into at least two disjoint, induced subgraphs. If the size of each is at most a constant fraction 1 of that of G, the separator is said to be *-balanced*. Lipton and Tarjan [LT79] and Miller [Mil86] showed that in any planar graph of n vertices,

Algorithm 2.2: GrahamScan(P)

input : An array P of n pseudolines, ordered with respect to their y-values at x 8

output: A stack S of the pseudolines of P that participate in the upper envelope of its arrangement

```
1  S:PUSH(P[0], S)
2  for i ← 1 to n − 1 do
3      while true do
4          if P[i] is and S:TOP do not intersect then
5              break
6          else
7              if S:NEXT-TOP = NULL then
8                  S:PUSH(P[i], S)
9                  break
10             else
11                 Let v be the intersection point of S:TOP and S:NEXT-TOP
12                 Let w be the intersection point of S:TOP and P[i]
13                 if w lies to the right of v then
14                     S:PUSH(P[i], S)
15                     break
16                 else
17                     S:POP
18 return S
```

we can compute a 2{3-balanced separator of $O(n^{?} \bar{n})$ size in linear time. The former [LT79, Lemma 2] also proved that given any spanning tree T in a triangulated planar graph, there are two paths R and Q that stem from the root of T, such that the removal of their vertices decomposes G into two disjoint planar subgraphs of $2n${3 vertices each. Moreover, R and Q can be computed in linear time [LT79, Steps 1, 8, and 9 in Section 3]. If T is a shortest-path tree, we say that C $R \cup Q$ is a *shortest-path separator* (notice though that the size of C could be as big as n).

In Chapter 5, we employ the following lemma based on [LT79] to compute a $(1+\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph and construct $(1+\varepsilon)$-approximate distance oracles for it.

Lemma 2.3. (Shortest-path separators in planar graphs) *Given a triangulated planar graph G and a shortest-path tree T, we can compute two paths R and Q that stem from the root of T in linear time, such that C $R \cup Q$ is a 2{3-balanced shortest-path separator.*

Given a planar graph, we can apply any -balanced shortest-path separator (where

1 is a constant), remove its vertices, and recursively handle every ensuing subgraph, thus producing a *decomposition tree* $\mathcal{T}$ with the following properties.

- Each node ν of $\mathcal{T}$ is associated with a subset $V^{\nu} \subseteq V$. The subsets V^{ν} over all children of ν are disjoint and contained in V^{ν}. If ν is the root, $V^{\nu} = V$. If ν is a leaf, V^{ν} has $O(1)$ size.
- Each non-leaf node ν of $\mathcal{T}$ is associated with a set of paths, called *separator paths*, which are classified as either "internal" or "external". The internal separator paths cover precisely all vertices of $V^{\nu} \setminus \bigcup_{\text{child } \mu \text{ of } \nu} V^{\mu}$, while the external are outside of V^{ν}.
- For each child μ of a non-leaf node ν, every neighbor of the vertices of V^{μ} in H is either in V^{ν} or in one of the (internal or external) separator paths at ν.

Similarly to Kawarabayashi, Sommer, and Thorup [KST13, Section 3.1], we can construct such a tree $\mathcal{T}$ in $O(n \log n)$ time with the extra properties that (i) its height is $O(\log n)$, and (ii) each non-leaf node ν has $O(1)$ separator paths. We apply these decomposition trees to solve $(1+\varepsilon)$-approximate shortest-path problems in weighted unit-disk graphs in Chapter 6.

2.6 Abstract Voronoi diagrams

Klein [Kle89] introduced the concept of *abstract Voronoi diagrams* to unify the treatment of various Voronoi diagrams in the plane. We use the more recent definition of Cabello [Cab17b, Section 4]. Let S be a set of *abstract sites*. The *abstract bisector* of each ordered pair (s, t) with $s, t \in S$ is a pair $(A(s, t), D(s, t))$ where $A(s, t)$ is a simple planar curve that bounds the open region $D(s, t)$. Intuitively, the points on $A(s, t)$ are equidistant (with respect to some distance function) to s and t, while the points in $D(s, t)$ are strictly closer to s than to t. The abstract Voronoi region of each $s \in S$ is defined as $\mathrm{AVR}(s, S) = \bigcap_{t \in S \setminus \{s\}} D(s, t)$. Then, the abstract Voronoi diagram of S is naturally defined as $\mathrm{AVD}(S) = \mathbb{R}^2 \setminus \bigcup_{s \in S} \mathrm{AVR}(s, S)$.

A system of abstract bisectors $\{(A(s, t), D(s, t)) \mid s, t \in S, s \neq t\}$ is *admissible* if it satisfies the following properties.

- For each $s, t \in S$ with $s \neq t$:
  - $A(s, t) = A(t, s)$ and
  - $\mathbb{R}^2$ is the disjoint union of $A(s, t)$, $D(s, t)$, and $D(t, s)$.
- There exists a special point $p_8 \in \mathbb{R}^2$ such that $A(s, t)$ for each $s, t \in S$ passes through it.

- For each set S^1 of three sites of S:
  - $\mathrm{AVR}(s, S^1)$ is path-connected for each $s \in S^1$ and
  - $\mathbb{R}^2 \setminus \bigcup_{s \in S^1} \overline{\mathrm{AVR}(s, S^1)}$, where $\overline{A}$ is the closure of $A \subseteq \mathbb{R}^2$.

For a system of admissible abstract bisectors, the corresponding abstract Voronoi diagram is a planar graph with a fixed embedding [KLN09, Theorem 10]. Thus, it is composed of *abstract Voronoi nodes* (of degree at least three) and *arcs*. Klein et al. [KMM93] showed how to construct such an abstract Voronoi diagram by extending a standard randomized incremental construction algorithm for Euclidean Voronoi diagrams [CS89, Mul94]. Their algorithm requires an expected number of $O(n \log n)$ or a worst-case number of $O(n^2)$ elementary operations, where n is the number of sites. An elementary operation therein is the computation of the abstract Voronoi diagram of any four sites. Moreover, in that algorithm each abstract Voronoi node (respectively arc) is represented with a pointer to an abstract Voronoi node (respectively arc) of an abstract Voronoi diagram of four sites. We use abstract Voronoi diagrams to compute a $(1+\varepsilon)$-approximation of the diameter of non-negatively-weighted, undirected planar graphs in Chapter 5.

2.7 Sparse neighborhood covers

The concept of *sparse neighborhood covers* was introduced by Awerbuch and Peleg [AP90] in the context of distributed computing. Given a graph $G = (V, E)$ and two integers and r, a $(, r)$-*sparse neighborhood cover* is a collection of subsets $V_1, V_2, \ldots, V_k$, for some $k \geq 0$, of V with the following properties:

- The subgraph of G induced by each V_i has radius at most $O(r)$.
- Each vertex $u \in V$ belongs in at most subsets V_i.
- For every $u \in V$, there is some V_i that contains each $v \in V$ with $dist_G(u, v) \leq r$.

If G is a planar graph of n vertices, Busch et al. [BLT07] showed how to construct such a cover with $O(1)$ (thus, of linear size). Kawarabayashi et al. [KST13] proved that the construction takes $O(n \log n)$ time. We employ sparse neighborhood covers in Chapters 5 and 6 to construct $(1+\varepsilon)$-approximate distance oracles for non-negatively-weighed, undirected planar graphs and for weighted unit-disk graph respectively.

2.8 Well-separated pair decompositions

The *well-separated pair decomposition* (WSPD) is a well-known technique in computational geometry, introduced by Callahan and Kosaraju [CK95] to address proximity problems in

the Euclidean (or any L_p) metric. We give here a definition of WSPD that holds not only for the Euclidean, but for any metric space (S, d), where S is a set of elements, and d is the distance function defined on $S \times S$. Let $diam(S') = \max_{s_1,s_2 \in S'} d(s_1, s_2)$, and let $d(S_1, S_2) = \min_{s_1 \in S_1, s_2 \in S_2} d(s_1, s_2)$, where $S', S_1, S_2 \subseteq S$. Two subsets A and B of S are called *c-well-separated* for some $c > 0$, if $\max\{diam(A), diam(B)\} \leq c \cdot d(A, B)$. See Figure 2.2 for an example.

Figure 2.2: A 2-well-separated pair of points in the Euclidean space.

Given a metric space (S, d) and a parameter c, a *c-well-separated pair decomposition* is a set of pairs $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$, where $P_i = (A_i, B_i)$, with the following properties.

- For each i,
  - $A_i, B_i \subseteq S$,
  - $A_i \cap B_i = \emptyset$, and
  - (A_i, B_i) is c-well-separated.
- For each $a, b \in S$, there is a unique i with $a \in A_i$ and $b \in B_i$.

For the Euclidean metric and any $c > 1$, Callahan and Kosaraju showed how to construct a c-WSPD of a linear number of pairs in near-linear time, sparking thus near-linear-time WSPD-based algorithms for many problems, such as k-nearest neighbors, N-body potential fields, geometric spanners, and approximate minimum spanning trees. See [CK95, Kle08] for more details.

Gao and Zhang [GZ05] extended the concept of WSPDs to the weighted unit-disk-graph metric (S, d), where each element of S is a vertex of a unit-disk graph G, and $d(\cdot, \cdot)$ corresponds to $dist_G(\cdot, \cdot)$. Specifically, they gave an $O(c^4 n \log n)$-time algorithm that constructs a c-WSPD of $O(c^4 n \log n)$ pairs under the new metric. By choosing two *representatives* $a \in A$ and $b \in B$ for each $(A, B) \in \mathcal{P}$ and computing $dist_G(a, b)$, the distance of any two vertices of G can be $(1+\varepsilon)$-approximated with the distance of one of these representative pairs. Gao and Zhang showed how to find that representative pair in $O(1)$ time and used that scheme to devise many shortest-path results, such as a $(1+\varepsilon)$-approximation algorithm for the diameter and $(1+\varepsilon)$-approximate distance oracles.

By setting $c = 1/\varepsilon$, we have the following lemma, which we use in Chapter 6 as a subroutine in our shortest-path solutions for weighted unit-disk graphs.

Lemma 2.4. (WSPD in unit-disk graphs) *Given a set S of n planar points, we can find a set of $O((1/\varepsilon)^4 n \log n)$ pairs of them in $O((1/\varepsilon)^4 n \log n)$ time, such that the shortest-path*

distance between any two vertices in the weighted unit-disk graph of S can be $p1 \quad "q$*-approximated by the shortest-path distance between one of these pairs, which can be found in* $O p1q$ *time.*

Chapter 3

Single-source and all-pairs shortest paths in unit-disk graphs

In this chapter, we study the single-source shortest paths (SSSP) and the all-pairs shortest paths (APSP) problem in unweighted unit-disk graphs of n vertices. We show that, after presorting the disk centers, we can solve SSSP from any source in linear time, thus improving upon the $O(n \log n)$-time algorithm of Cabello and Jejčič [CJ15]. Moreover, we give the first slightly-subquadratic-time APSP algorithm, running in $O(n^2 \frac{\log\log n}{\log n})$ time. Specifically, our algorithm computes an implicit representation of all pairwise shortest paths. In the same amount of time, we can also compute the diameter of the graph.

The results of this chapter have been presented in ISAAC 2016 [CS16].

Definitions. Recall that an unweighted unit-disk graph G is the intersection graph of a set of unit-diameter disks in the plane. That is, vertices correspond to a set S of planar points, namely the centers of the disks, and there is an unweighted edge between every two points of S at Euclidean distance at most one. We assume that G is represented implicitly by S (thus, only linear space is required to store it) and that without loss of generality it is connected.

Let S be a set of planar points, and let G be the unweighted unit-disk graph they define. Then, for any $s; t \in S$, we denote a s-to-t shortest path in G by $\pi(s; t)$ and its length by $dist(s; t)$. We also refer to $dist(s; t)$ as shortest-path distance or simply distance. Let $pred(s; t)$ be t's predecessor on $\pi(s; t)$. Also, we define the shortest-path tree $T(s)$ of $s \in S$ to be a spanning tree of G rooted at s, such that for each $t \in S$, the s-to-t shortest-path distance in $T(s)$ corresponds to $dist(s; t)$. We are interested in the following fundamental shortest-path problems in G.

- The *single-source shortest paths* (SSSP) problem, i.e., given a source $s \in S$, compute $dist(s; t)$ and $pred(s; t)$ for each $t \in S$.

- The *all-pairs shortest paths* (APSP) problem, i.e., compute $dist(s,t)$ and $pred(s,t)$ for each $s,t \in S$ or an implicit representation that supports their retrieval in $O(1)$ time.

Background and overview of techniques. Roditty and Segal [RS11] used the ideas of Chan and Efrat [CE01] to reduce SSSP in unweighted unit-disk graphs to dynamic nearest-neighbor searching. Therefore, by employing the data structure of Chan [Cha10a], they obtained an $O(n \log^6 n)$-time algorithm. Cabello and Jejčič [CJ15] solved SSSP in $O(n \log n)$ time by first computing the Delaunay triangulation of the given point set and then repeatedly performing nearest-neighbor queries. Thus, their approach needs $\Omega(n \log n)$ time even after presorting. According to [CJ15], Efrat has also observed an alternative grid-based $O(n \log n)$-time algorithm, but his suggested solution used Efrat et al.'s [IK01] semi-dynamic data structure, which also inherently requires $\Omega(n \log n)$ time even excluding presorting. Instead, we present a simple method that solves SSSP from any source in linear time, after presorting the given point set. Our algorithm uses a grid-based approach and exploits a linear-time Graham-scan-like procedure [PS85] for computing upper envelopes of unions of unit disks that are presorted with respect to their centers.

Then, as an intermediary step for our slightly-subquadratic-time APSP algorithm, we extend our SSSP algorithm to the *multiple-sources shortest paths* (MSSP) problem. In MSSP we want to find the shortest-path tree of *many* sources that lie in a *cluster*, i.e., in a common grid cell. Now we have to compute not just one but many upper envelopes, one for each source, which we formalize as a *preprocessed universe* problem. That is, preprocess a set of unit disks (a "universe"), such that given any subset of that universe, we can compute its upper envelope in slightly *sublinear* time. Note that the input subset and the output can be encoded with a linear number of bits, thus with a slightly sublinear number of words. We provide a solution by properly using bit tricks and lookup tables, hence making an unusual addition to the relatively recent research direction of preprocessed universe problems (e.g., see [CM11, BM11, EM13, CL15]).

We finally provide the first slightly-subquadratic-time APSP algorithm for unweighted unit-disk graphs, by drawing inspiration from the planar graphs case [Cha12, WN13b]. Therein, balanced separators[1] are used to decompose the given graph into regions of polylogarithmic size, each then handled with table-lookup techniques. Unfortunately, that approach cannot be applied directly in a unit-disk graph because for one thing such a graph might contain large cliques. However, we show that we can address cliques with our MSSP algorithm and replace separators with a simple shifted-grid strategy [HM85], which is standard for geometric approximation algorithms but here is used to design an *exact* solution.

[1] A balanced separator of a graph H is a subset of its vertices whose removal separates it into a small number of vertex disjoint subgraphs, such that the size of each is at most a constant fraction of that of H.

3.1 SSSP in linear time after presorting

In this section we show that given a set S of n planar points, which have been presorted with respect to their x- and y-coordinates, we can compute the shortest-path tree from any $s \in S$ in the weighted unit-disk graph of S in linear time. First, we construct a uniform grid of square cells, each of side length $\frac{1}{\sqrt{2}}$. We call two cells c and c' *neighbors* if and only if their minimum Euclidean distance is at most one. Thus, the number of neighbors of a cell is upper-bounded by a constant. See Figure 3.1.

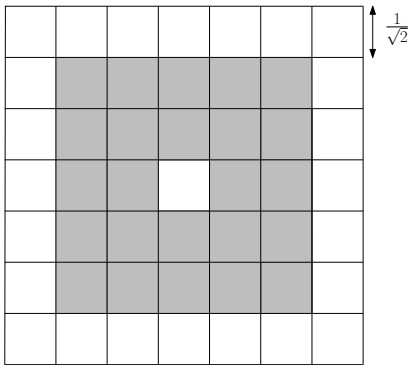


Figure 3.1: A uniform 8 × 8 grid and the neighbors of the cell in the fourth row and column (grey).

Then, we devise a BFS-like approach, similar to that in Section 2.3. See Algorithm 3.1 for the pseudocode. Specifically, we perform $n - 1$ steps, where in the beginning of each step ℓ, we assume that we have properly processed each $t \in S$ at distance at most $\ell - 1$ from s. We call the rest of the points *undiscovered* and those at distance exactly $\ell - 1$ from s *frontier*. As in classical BFS, we want to find every undiscovered point that shares an edge with a frontier point, but inspecting all edges incident to the latter ones would require quadratic total time. However, in unit-disk graphs an edge exists only between points at Euclidean distance at most one. Hence, it suffices to consider each pair (c, c'), where c is a cell that contains frontier points and c' a neighbor of c, and find every undiscovered point t in $S \cap c'$, such that there exists a frontier point z in $S \cap c$ with $\|z - t\| \leq 1$ (Line 10 in Algorithm 3.1). To do so, we would like to solve the following subproblem.

Subproblem 3.1. (Intersection detection of unit disks in linear time) *Consider a set of n_r red points below a horizontal line h and another set of n_b blue points above h, both presorted by x. Find for each blue point a red point at Euclidean distance at most one from it (if any). The total time should be linear.*

In our application, the red points are the frontier points of $S \cap c$, the blue points are the undiscovered points of $S \cap c'$, and h is any horizontal line that separates c from c'. Since c and c' are either horizontally or vertically separated, we assume the former without loss of generality.

Algorithm 3.1: UnitSSSP($S; s$)

build a uniform grid of of square cells, each of side length $\frac{1}{\sqrt{2}}$
$distrs; ss \quad 0$
$distrs; ts \quad 8, @t \in S \quad tsu$
$predrs; ts \quad NULL, @t \in S$
$F \quad tsu$
for $` \quad 1$ *to* $n \quad 1$ **do**
  $F^1 \quad ?$
  for *each grid cell* c *that contains a frontier point* **do**
    for *each neighbor* c^1 *of* c **do**
      for *each undiscovered point* $t \in S \cap c^1$, *s.t.* $\exists$ *frontier point* $z \in S \cap c$ *with* $\}z \quad t\} \leq 1$ **do**
        $distrs; ts \quad `$
        $predrs; ts \quad z$
        $F^1 \quad F^1 \cup ttu$
  $F \quad F^1$
return $distrs;$ s *and* $predrs;$ s

Lemma 3.1. (Intersection detection of unit disks in linear time) *We can solve Subproblem 3.1 in* $O(n_r \quad n_b)$ *time.*

Proof. Consider a unit disk centered at each red point, and let U be the part of the *upper envelope* of these disks (i.e., the boundary of their union) that lies above h. Then, a blue point is at Euclidean distance at most one from any red point if and only if it lies below U. See Figure 3.2(a)-(c).

Consider the part of the boundary of each unit disk that lies above h. There is at most one intersection between any two such parts (and can be found in constant time), so we can add two line segments at the endpoints of each, such that no intersections take place below h, to obtain a *pseudoline family* as defined in Section 2.4. Furthermore, we can ensure that the y-order of the pseudolines at $x \quad 8$ coincides with the x-order of the red points, which are presorted. See Figure 3.2(d)-(f). Thus, we can compute the upper envelope U^1 of that family in linear time with the Graham-scan-like algorithm of Theorem 2.2, as in Section 2.4. Since U^1 corresponds to U above h, with a linear scan we can find for each blue point whether it is below U^1, and if so, return a red point whose unit disk contains it. □

Crucial to upper-bounding the running time of our algorithm is the number of steps a cell is *visited*, i.e., we perform an operation on any of its points.

Lemma 3.2. (Cell visits in SSSP) *Each cell is visited only a constant number of times.*

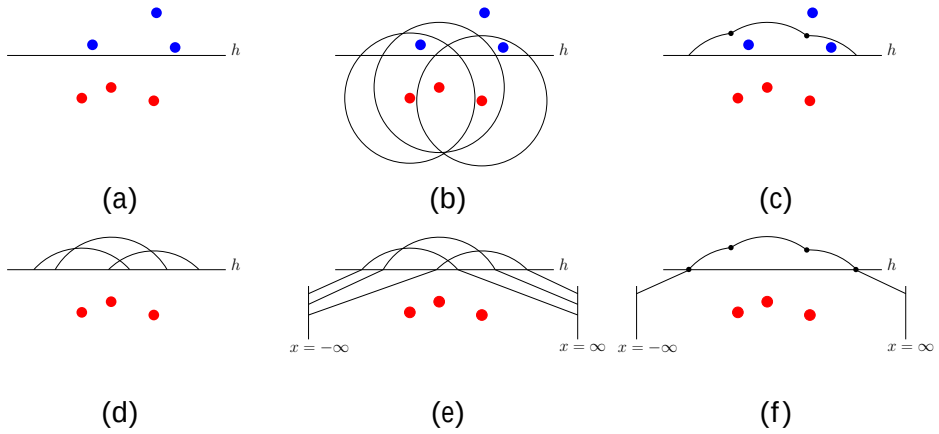


Figure 3.2: (a), (b), (c) A set of red and blue points, the unit disks centered at the former, and the part of the upper envelope U of these disks above h. (d), (e), (f) The part of each unit disk above h, the corresponding pseudoline family, and its upper envelope U^1.

Proof. A cell c is visited whenever a point of either (i) c or (ii) a neighbor c^1 of c is in the frontier. The diameter of c is one, so the first time any of its points enters the frontier, the rest will do so either in the same or in the next step. Thus, c is visited in case (i) at most twice because each point is in the frontier exactly once. That, along with the fact that c has $O(1)$ neighbors, implies that c is visited $O(1)$ times in case (ii) as well. □

New Result 1. (SSSP and APSP in unweighted unit-disk graphs) *We can preprocess an unweighted unit-disk graph of n vertices in $O(n \log n)$ time, such that we can solve SSSP for any given source in linear time. Consequently, we can solve APSP in quadratic time.*

Proof. In unit-disk graphs, there is an edge between two points if and only if their Euclidean distance is at most one, so our algorithm derives its correctness from that of BFS. Initially assigning points to grid cells can be done in linear time (without hashing because of presortedness), so by Lemmas 3.1 and 3.2, the total time of our algorithm is also linear. Applying our algorithm once per vertex, we can also solve APSP in quadratic time. □

3.2 Multiple-sources shortest paths in linear time

Now we extend our SSSP algorithm of Section 3.1 for the multiple-sources shortest paths (MSSP) problem, where we want to compute the shortest-path trees of multiple sources $s_1, \ldots, s_k$ for some parameter k to be determined later. We assume that all sources lie in a cluster, i.e., in the same grid cell. The MSSP algorithm we describe here will be used by our slightly-subquadratic-time APSP algorithm in Section 3.3. Our approach to solve MSSP, inspired by the APSP result of Chan [Cha12, Section 3], is to simultaneously run a BFS-like algorithm for all k sources, but avoid a factor-k slowdown by using bit-packing tricks.

Each $t \in S$ is now associated with k distances, so we maintain a vector $dist[t]$ that contains $dist(s_1, t)$ and $dist(s_i, t) - dist(s_1, t)$ for each source s_i. Since all sources lie in a unit-diameter square, the triangle inequality implies that these differences are in $[-1, 0, 1]$. Hence, $dist[t]$ can be encoded with $O(\log n + k)$ bits. We will show later how to encode t's predecessors (in Lemma 3.6).

We perform $n - 1$ steps, where in each step ℓ and for each source s_i, we assume that we have properly processed each $t \in S$ at distance at most $\ell - 1$ from s_i. We call the rest of the points s_i-*undiscovered* and those at distance exactly $\ell - 1$ from it s_i-*frontier*. Thus, throughout the algorithm we maintain for each $t \in S$, a k-bit vector $frontier[t]$, where the i-th bit denotes whether t is an s_i-frontier point, and another k-bit vector $undiscovered[t]$, similarly defined.

As in our SSSP algorithm, in every step ℓ we consider each cell c that has frontier points (for any source). First, we construct a bit vector $frontier_c[s_i]$ for each source s_i, where the j-th bit denotes whether the j-th point in the x-ordered sequence of the points of c is s_i-frontier. Then, we want to find a k-bit vector $output_{c,c'}[s_i]$ for each neighbor c' of c, where the j-th bit denotes whether the j-th point in the x-ordered sequence of the points of $S \times c'$ is at Euclidean distance at most one from any s_i-frontier point in $S \times c$. We also want to compute a data structure $pred_{c,c'}[s_i]$ that supports the following kind of queries: given any point $t \in S \times c'$, find in constant time t's predecessor on the s_i-to-t shortest path if it lies in $S \times c$. See Algorithm 3.2 for the pseudocode.

Algorithm 3.2: UnitMSSP$(S, s_1, s_2, \ldots, s_k)$

```
1  build a uniform grid of square cells, each of side length 1/√2
2  for t ∈ S do
3      initialize dist[t], frontier[t], undiscovered[t]
4  for ℓ ← 1 to n − 1 do
5      for each grid cell c that contains a frontier point for any source do
6          for each source s_i do
7              build frontier_c[s_i]
8          for each neighbor c' of c do
9              build output_{c,c'}[s_i]
10             build pred_{c,c'}[s_i]
11             for each point t of c' do
12                 update dist[t], frontier[t], undiscovered[t]
13 for each source s_i do
14     merge pred_{·}[s_i] into a single data structure pred[s_i]
15 return dist[s_1], ..., dist[s_k] and pred[s_1], ..., pred[s_k]
```

We now show how to build each vector $frontier_c[\cdot]$ in linear total time (Line 7 in Algorithm 3.2). We can similarly use $output_{c,c'}[\cdot]$ to update $dist[\cdot]$, $frontier[\cdot]$, and $undiscovered[\cdot]$ (Line 12 in Algorithm 3.2).

Lemma 3.3. (Frontier points) *Given a cell c and a parameter g with $kg \leq \log n$, we can construct $\mathit{frontier}_c(s)$ in linear total time.*

Proof. We divide the points of c into chunks of g consecutive points each, collect the g k-bit vectors $\mathit{frontier}(p_1), \ldots, \mathit{frontier}(p_g)$ of the points $p_1, \ldots, p_g$ of each chunk j, and *transpose* them to generate k g-bit vectors $\mathit{fro}_j(s_1), \ldots, \mathit{fro}_j(s_k)$, where the z-th bit of $\mathit{fro}_j(s_i)$ is equal to the i-th bit of $\mathit{frontier}(p_z)$. For each source s_i, we concatenate $\mathit{fro}_1(s_i), \mathit{fro}_2(s_i), \ldots$ to generate $\mathit{frontier}_c(s_i)$. The transposition involves merely shuffling bits between words and can be straightforwardly implemented in constant time with table lookup if $kg \leq \alpha \log n$, where $0 < \alpha < 1$ is a constant, after $O(n)$ preprocessing time as explained in Section 2.1. □

To build the vectors $\mathit{output}_{c,c^1}(s)$ (Line 9 in Algorithm 3.2), we cannot apply the linear-time approach of Lemma 3.1 for each source because that would lead to $O(nk)$ total time. Instead, we would like to solve k instances of an extended version of Subproblem 3.1 in linear total time.

Subproblem 3.2. (Intersection detection of unit disks in sublinear time) *Consider a universe R of n_r red points below a horizontal line h and another universe B of n_b blue points above h, both presorted by x. Preprocess R and B such that given any subset $Q \subseteq R$, we can determine for each blue point whether there is a red point of Q at distance at most one from it in* sublinear *total time.*

By sublinear here we mean $O\left(\frac{n_r + n_b}{\mathrm{polylog}\, n}\right)$, where $O(\log n)$ is the word size. Note that we can represent (a) the input subset Q as an n_r-bit vector, where the j-th bit denotes whether the j-th red point is in Q, and (b) the output as an n_b-bit vector, where the j-th bit denotes whether the j-th blue point is at distance at most one from a red point in Q. Thus we can pack the two vectors with $O\left(\frac{n_r}{\log n}\right)$ and $O\left(\frac{n_b}{\log n}\right)$ words respectively.

In our case, the red points are the s_i-frontier points in c and the blue ones are all points of c^1, while the input and output vectors are $\mathit{frontier}_c(s_i)$ and $\mathit{output}_{c,c^1}(s_i)$, respectively. As we have seen in the proof of Lemma 3.1, the key to solving Subproblem 3.1 lies in constructing upper envelopes, so we first focus on the following subproblem.

Subproblem 3.3. (Upper envelopes of unit disks in slightly sublinear time) *Consider a universe R of n_r red points below a horizontal line h, presorted by x. Preprocess R such that given any subset $Q \subseteq R$, we can compute the upper envelope of the unit disks of the red points of Q (specifically, the part above h) in* sublinear *time.*

Note that again we can represent the input and the output with a sublinear number of words, namely $O\left(\frac{n_r}{\log n}\right)$. Specifically, the input is represent as above and the output as a n_r-bit vector, where the j-th bit denotes whether the unit disk of the j-th red point participates in the relevant upper envelope.

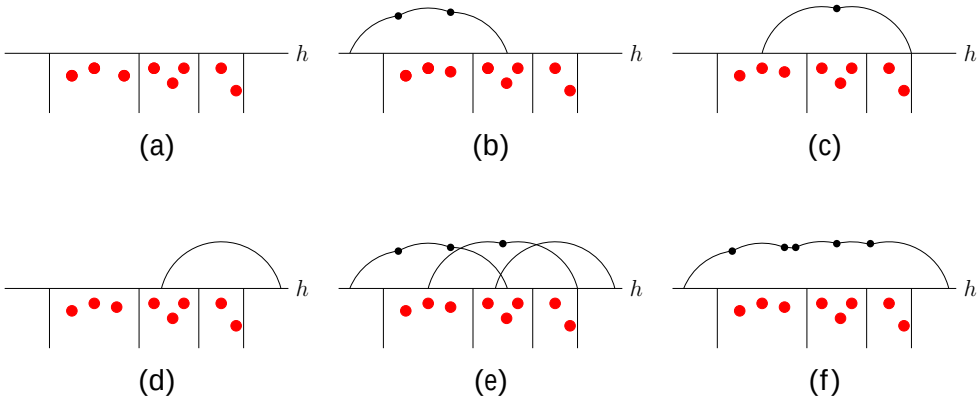


Figure 3.3: (a) The chunks of eight points for $g = 3$. (b), (c), (d) The small upper envelope of all points of each chunk. (e) The three small upper envelopes superimposed. (f) The upper envelope of the small upper envelopes.

Lemma 3.4. (Upper envelopes of unit disks in slightly sublinear time) *We can solve Subproblem 3.3 with $O(n_r \log n_r)$ $n_r 2^g$ preprocessing and $O(n_r \frac{\log g}{g})$ query time for any given $g \in [\log n]$.*

Proof. In the preprocessing phase, we divide the red points into $O(\frac{n_r}{g})$ chunks, which lie in $O(\frac{n_r}{g})$ disjoint slabs (regions bounded by two vertical lines) of at most g consecutive points each. For each red point, consider a unit disk centered at it. For each of the $O(2^g)$ possible subsets of the points of each chunk, we compute the upper envelope of the relevant unit disks (specifically, the part that lies above h), called *small upper envelope*, in $O(g)$ time as in the proof of Lemma 3.1. We represent each with a g-bit vector and store all such vectors in a lookup table. The preprocessing time is $O(n_r 2^g)$. See Figure 3.3.

Given a subset of the red points, we can compute the upper envelope U of their unit disks (specifically, the part that lies above h) by merging the small upper envelopes of the relevant subset of each chunk. Since the chunks are vertically separated, any two of these $O(\frac{n_r}{g})$ small upper envelopes intersect each other at most once. Thus, similarly to the proof of Lemma 3.1, we can view the family of small upper envelopes as a family of pseudolines and apply the Graham-scan-like procedure of Theorem 2.2 to compute the upper envelope U^1 of their arrangement. We can find the intersection point between two small upper envelopes with binary search in $O(\log g)$ time as in [OvL81], so U^1 can be constructed in $O(\frac{n_r}{g} \log g)$ time. In $O(\frac{n_r}{\log n})$ additional time, we can create the desired bit-vector representation for U. □

Lemma 3.5. (Intersection detection of unit-disks in slightly sublinear time) *We can solve Subproblem 3.2 with $O(n_r 2^g g + n_b g)$ preprocessing and $O(n_r + n_b \frac{\log g}{g})$ query time for any given $g \in [\log n]$.*

Proof. We build upon the method in the proof of Lemma 3.4. Specifically, we first perform the steps of the preprocessing phase therein and then divide the blue points into $O(\frac{n_b}{g})$ chunks, which lie in that many disjoint slabs of at most g consecutive points each. See Figure 3.4(a). For the rest of the proof, any reference to small upper envelopes is to those constructed in the proof of Lemma 3.4, and any reference to slabs is to those that correspond to the blue points. We store the following extra structures.

1. For each small upper envelope e and each slab σ that contains at least one of its vertices, we compute a g-bit vector where the j-th bit denotes whether the j-th blue point in σ is below e. We store all these vectors in a lookup table. There are $O(\frac{n_r}{g} 2^g)$ small upper envelopes each with $O(g)$ vertices, so the total time for this step is $O(n_r 2^g g)$. See Figure 3.4(b).

2. For each slab σ, consider the unit disks centered at each blue point therein. We compute the $O(g^2)$-complexity arrangement of these $O(g)$ disks and then build a *point location* structure [PS85] for it in $O(g^2)$ time. Moreover, we record for each face of the arrangement a g-bit vector where the j-th bit denotes whether that face is inside the unit disk of the j-th blue point. There are $O(\frac{n_b}{g})$ slabs, so this step takes $O(n_b g)$ time. See Figure 3.4(c).

To answer a query, we first construct the upper envelope E of the $O(\frac{n_r}{g})$ relevant small upper envelopes as described in the proof of Lemma 3.1. We determine, for each blue point, whether it is below E by scanning the slabs from left to right. Consider the next slab σ and each small upper envelope e that contributes arcs to E in σ. We compute a bit vector $\mathbf{v}_{\sigma,e}$ where the j-th bit denotes whether the j-th blue point in σ is below e, as follows.

1. If σ contains at least one vertex of e, then $\mathbf{v}_{\sigma,e}$ has already been precomputed and lies in the lookup table of σ.

2. If σ contains no vertices of e, then only one unit disk contributes arcs to e inside σ. Let q be the center of that disk. We can determine $\mathbf{v}_{\sigma,e}$ in $O(\log g)$ time by first querying the point location data structure of σ with q and then finding the bit vector of that face in the lookup table of σ.

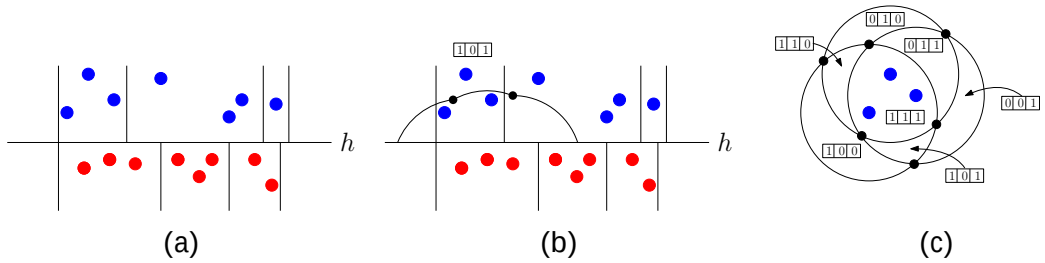


Figure 3.4: (a) The chunks of eight red and seven blue points. (b) The small upper envelope e of the unit disks of all red points of the (bottom) leftmost chunk and the bit vector for the vertex of e in the (top) leftmost slab. (c) The arrangement of the unit disks of the blue points of the (top) leftmost slab and the bit vectors of its faces.

Finally, we take the bitwise-or of the bit vectors $\mathbf{v}_{;e}$ over each small upper envelope e that contributes to E in . The total number of bitwise-or operations and point location queries is $O\left(\frac{n_r \; n_b}{g}\right)$, yielding $O\left(\frac{n_r \; n_b}{g} \log g\right)$ total query time. □

As explained earlier, we want to query the data structure of Lemma 3.5 with each source s_i to build $output_{c;c^1}[s_i]$ (Line 9 in Algorithm 3.2) in linear total time. Thus, by setting $g \quad k \log k$, we have that these queries can be performed in $O\left(k(n_r \quad n_b)\frac{\log g}{g}\right)$ $O(n_r \quad n_b)$ total time.

To implement the predecessor data structure $pred_{c;c^1}[s_i]$ for each source s_i (Line 10 in Algorithm 3.2), we need to augment the data structure of Lemma 3.5 to also find, for each blue point, one *witness* (if it exists), i.e., a red point at Euclidean distance at most one from it.

Lemma 3.6. (Predecessors) *We can extend the data structure in Lemma 3.5, such that given any blue point, we can report one* witness *(if it exists) in constant time.*

Proof. We build upon the proof of Lemma 1. First, in the lookup tables we build in the preprocessing phase therein (Step 1), we also record witnesses for the true bits of each g-bit vector, which we store in a $(g \log g)$-bit *witness vector*. Consider a query and a slab and remember that the upper envelope E constructed in that query consists of pieces of small upper envelopes. We divide each slab (of the blue points) into subslabs by drawing vertical lines at the endpoints of these pieces. Then, we mark the rightmost blue point b of each subslab, which can be found by binary searches in $O\left(\frac{n_r \; n_b}{g} \log g\right)$ total time. If a small upper envelope e participating in E has vertices in a subslab of , we create a pointer from b to $\mathbf{v}_{;e}$, else to the red point whose unit disk contributes the only arc of e inside . For each non-empty subslab, we also create pointers for its blue points to it.

Given any blue point q (assuming without loss of generality that it has a witness), we can retrieve the pointer to its subslab , find the pointer of the marked blue point b of q, and then follow b's pointer to the small upper envelope e in . If e has at least one vertex

in , we then look up q's witness with respect to $\mathbf{v}_{;e}$. Otherwise, we return the pointer of b to the red point that contributes the only arc of e in . □

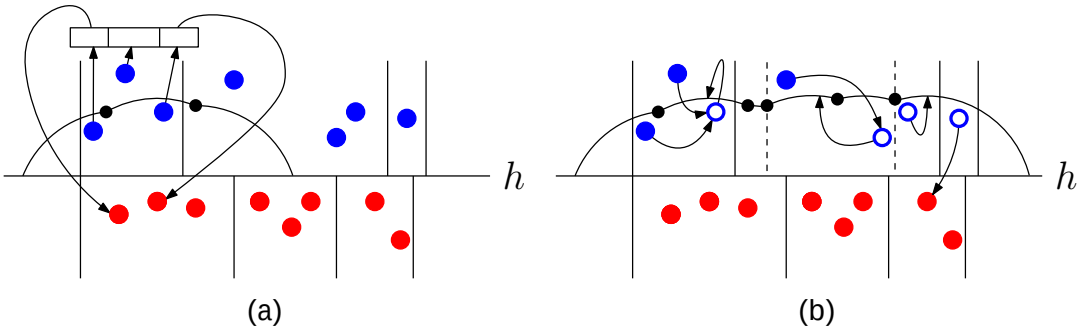


Figure 3.5: (a) The witness vector for the leftmost slab of blue points and the small upper envelope of all red points of the (bottom) leftmost chunk. (b) An upper envelope, the marked blue points, the subslabs, and the corresponding pointers (pointers to $\mathbf{v}_{;e}$ are shown as pointers to e).

It is straightforward to merge $pred_{;}(s_i)$ of each source s_i into a data structure $pred(s_i)$, such that the predecessor of any $t \in S$ in the s_i-to-t shortest path can be found in constant time.

Now we extend Lemma 3.2 for the case of multiple sources. The definition of *visiting* a cell is the same as in Section 3.1.

Lemma 3.7. (Cell visits in MSSP) *If the k source points are in the same cell, our algorithm visits each cell a constant number of times.*

Proof. For any two sources s and s' in the same cell, we know from the triangle inequality that $dist_s(p)$ and $dist_{s'}(p)$ can differ by at most one. Thus, the first time a point of a cell enters the frontier of any source, it will enter the frontier of the rest of the sources either in the same or in the next step of the algorithm. The rest of the proof is as the one of Lemma 3.2. □

We have $kg(n) = \log n$ and $g(k) = k \log k$, so we set $k = O\left(\frac{\log n}{\log \log n}\right)$. Hence, we obtain the following theorem, whose its correctness stems from that of BFS.

Theorem 3.8. (Multiple-sources shortest paths in unweighted unit-disk graphs) *We can preprocess, in $O(n \log n + n2^{O(k \log k)})$ time, an unweighted unit-disk graph of n vertices, such that we can compute an implicit representation of the shortest-path trees from any $k = O\left(\frac{\log n}{\log \log n}\right)$ source points that lie in a cluster of unit diameter in linear time.*

By implicit representation, we mean that we can retrieve $dist(s;t)$ and $pred(s;t)$ for any $s;t \in S$ in constant time.

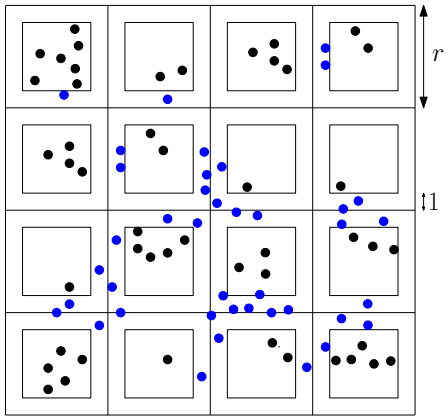


Figure 3.6: A shifted grid (boundary points in blue).

3.3 APSP in slightly subquadratic time

We finally present our slightly-subquadratic-time APSP algorithm for unweighted unit-disk graphs. First, we build a grid composed of square cells, called *supercells*, with side length r, where $r \geq \frac{1}{\sqrt{2}}$ is a parameter to be specified later. We say that a point $p \in S$ is a *boundary point* if it is at Euclidean distance at most one from the boundary of some supercell. See Figure 3.6. To find such a translation with only a few number of boundary points, we use the standard *shifted* grid strategy of Hochbaum and Maass [HM85].

Lemma 3.9. (Shifted grid) *We can find a translation of S in $O(nr^2)$ time, such that the number of boundary points is $O(\frac{n}{r})$.*

Proof. First, we build the grid and then we shift the points of S by a random vector $\in R = [0,r)^2$. The probability that a point $s \in S$ is a boundary point after shifting is equal to that of lying in two fixed rows or columns in R, which is upper-bounded by $\frac{4}{r}$. Hence, the expected number of boundary points is $O(\frac{n}{r})$.

To derandomize, we can try every vector in $r \in R$, count the number of boundary points of the ensuing translation of S, and return $\arg\min_{\in R}$ in $O(nr^2)$ total time. □

Then, our algorithm proceeds in four steps.

1. We first compute the shortest paths from the $O(\frac{n}{r})$ boundary points in $O(\frac{n^2}{r})$ total time with our linear-time SSSP algorithm of Theorem 1.

2. Next, for each supercell $_i$ that contains more than a points or more than b boundary points, where a and b are parameters to be set later, we compute the shortest paths from all points in $S \cap _i$ with our MSSP algorithm of Section 3.2 as follows.

First, we decompose $_i$ into $O(r^2)$ cells of diameter one and then group its points into $O\left(\frac{|S \cap _i|}{k}\right) \quad r^2$ clusters of size at most k. The number of supercells with more than a points is $O\left(\frac{n}{a}\right)$, and the number of supercells with more than b boundary points is $O\left(\frac{n}{rb}\right)$. Hence, the total time for this step is $O\left(_i \frac{|S \cap _i|}{k} \right) \quad r^2 \quad n \quad O\left(\frac{n}{k} \quad \frac{n}{a} \quad \frac{n}{rb}\right) r^2 \quad n \quad O\left(\frac{n^2}{k} \quad \frac{n^2 r^2}{a} \quad \frac{n^2 r}{b}\right)$.

3. For each supercell $_i$ with at most a points and at most b boundary points, we compute the shortest paths between all pairs of points in $S \cap _i$. We do so with a naive cubic-time APSP algorithm on $S \cap _i$, after adding an extra weighted edge between each boundary point u in $_i$ and each point $p \in S \cap _i$, with weight $dist_u(p)$, which we have computed in Step 1. These extra edges take care of the possibility that the shortest paths may not stay inside $_i$. The total time for this step is $O\left(_i |S \cap _i|^3\right) \quad O\left(_i |S \cap _i| a^2\right) \quad O(na^2)$.

4. Last, for each supercell $_i$ with at most a points and at most b boundary points, we compute an implicit representation of the shortest path between each $p \in S \cap _i$ and $q \in S \quad _i$. Such a path must pass through a boundary point in $_i$, so we want to find $\min_u \{dist(u,p) \quad dist(u,q)\}$ over all boundary points u in $_i$.

   We describe a table lookup method inspired by the APSP algorithm for planar graphs in [Cha12]. Specifically, for each connected component of the unit-disk graph of $S \cap _i$, we pick a *representative* boundary point $rep(p)$ (if one exists). Then, for each $q \in S \quad _i$, we define $signature(q)$ to be the vector that contains $dist(u,q) \quad dist(rep(u),q)$ over all boundary points u in $_i$. Notice that we have computed these values in Step 1. The distance between each u and $rep(u)$ is at most $O(r^2)$, so from the triangle inequality, $dist(u,q) \quad dist(rep(u),q)$ is bounded by $O(r^2)$. Hence, each $signature(q)$ can be encoded with $O(b \log r)$ bits, and we can generate them all in $O\left(\frac{n^2}{r}\right)$ total time.

   Let $signature_u(q) \quad dist(u,q) \quad dist(rep(u),q)$. Then, for each $p \in S \cap _i$ and each $q \in S \quad _i$, we have that

$$\min_u \{dist(u,p) \quad dist(u,q)\} \quad dist(rep(p),q) \quad \min_{u:rep(u) \quad rep(p)} \{dist(u,p) \quad signature_u(q)\}.$$

   For each $p \in S \cap _i$ and each possible signature, we can precompute the minimum in the right-hand side and store all of them in a lookup table in $|S \cap _i| 2^{O(b \log r)}$ time, for a total of $n 2^{O(b \log r)}$ time. Then, given $p \in S \cap _i$ and $q \in S \quad _i$, we can find $dist(s,t)$ and $pred(s,t)$ with table lookup in constant time.

The running time of the entire algorithm is

$$O\left(nr^2 \quad n \log n \quad n 2^{O(k \log k)} \quad \frac{n^2}{r} \quad \frac{n^2}{k} \quad \frac{n^2 r^2}{a} \quad \frac{n^2 r}{b} \quad na^2 \quad n 2^{O(b \log r)}\right).$$

To balance the fourth and the fifth term, we set $k = r$. For the fourth and the seventh, we set $r = \sqrt{b}$, so the running time becomes

$$O\left(nb + n \log n + n2^{O(\sqrt{b} \log b)} + \frac{n^2}{\sqrt{b}} + \frac{n^2 b}{a} + na^2\right).$$

Thus, by setting $a = b^{3/2}$ and $b = \frac{\varepsilon \log n}{\log \log n}$, where $\varepsilon > 0$ is a small constant such that the second and the eighth terms get absorbed by the others, we obtain $O\left(n^2 2^{\sqrt{\frac{\log \log n}{\log n}}}\right)$ total time. Our algorithm can retrieve any shortest-path distance in constant time and any shortest path in time proportional to its size.

To compute the diameter, we change Step 4 as follows. For each $p \in S \cap X_i$ and each possible signature, we now find the point $q \in S_i$ whose signature maximizes $dist_{rep}(p, q) + s$. All these maximums can be computed in $O\left(\frac{n^2}{r}\right)$ total time by scanning the distance values from all boundary points.

New Result 2. (APSP in unweighted unit-disk graphs) *We can compute an implicit representation of the shortest paths of all pairs of vertices and the diameter of an unweighted unit-disk graph of n vertices in $O\left(n^2 2^{\sqrt{\frac{\log \log n}{\log n}}}\right)$ time.*

Chapter 4

Single-source and all-pairs shortest paths in geometric intersection graphs

We present in this chapter a simple and general algorithm for the all-pairs shortest paths (APSP) problem in unweighted geometric intersection graphs. Specifically we reduce the problem to the design of static data structures for offline intersection detection. Consequently we can solve APSP in unweighted intersection graphs of n arbitrary disks in $O(n^2 \log n)$ time, axis-aligned line segments in $O(n^2 \log\log n)$ time, arbitrary line segments in $O(n^{7/3} \log^{1/3} n)$ time, d-dimensional axis-aligned unit hypercubes in $O(n^2 \log\log n)$ time for $d \geq 3$ and $O(n^2 \log^{d-3} n)$ time for $d \geq 4$, and d-dimensional axis-aligned boxes in $O(n^2 \log^{d-1.5} n)$ time for $d \geq 2$.

The results of this chapter have been presented in WADS 2017 [CS17a].

Definitions. Recall that an unweighted, undirected geometric intersection graph G is the intersection graph of a set S of geometric objects. That is, vertices correspond to objects and edges to pairwise intersections. We assume that we can determine in constant time if any two objects intersect each other, that G is represented implicitly by S (thus it can be stored with only linear space), and that G, without loss of generality, is connected.

Let S be a set of objects, and let G be the unweighted, undirected geometric intersection graph they define. Then, for any $s, t \in S$, we denote a s-to-t shortest path in G by $\pi(s, t)$ and its length by $dist(s, t)$. We also refer to $dist(s, t)$ as shortest-path distance or simply distance. Let $pred(s, t)$ be t's predecessor on $\pi(s, t)$. Also, we define the shortest-path tree $T(s)$ of $s \in S$ to be a spanning tree of G rooted at s, such that for each $t \in S$, the s-to-t shortest-path distance in $T(s)$ corresponds to $dist(s, t)$. We want to solve the following classical shortest-path problems in G.

- The *single-source shortest paths* (SSSP) problem, i.e., given a source $s \in S$, compute $dist(s,t)$ and $pred(s,t)$ for each $t \in S$ (equivalently, compute $T(s)$).
- The *all-pairs shortest paths* (APSP) problem, i.e., compute $dist(s,t)$ and $pred(s,t)$ for each $s,t \in S$ (equivalently, compute $T(s)$ for each $s \in S$).

Background and overview of techniques. We provide a simple and general APSP algorithm by a reduction to static, offline intersection *detection*: given a query object, decide whether there is an input object that intersects it (and report one if the answer is yes). First, given the shortest-path tree from a source vertex s as a guide, we show how to generate the shortest-path tree from an adjacent source vertex s' quickly. To do so, we exploit the fact that distances from s' are approximately known up to 1 and employ the right geometric data structures. Then, we generate the shortest-path trees of all vertices by visiting them in an order prescribed by a spanning tree. Some form of this simple idea has appeared before for general graphs (e.g., see [ACIM99, Cha12]), but it has been somehow overlooked by previous researchers in the context of geometric APSP.

Our solution compares favorably with the two previous general methods for the problem. The first runs an SSSP algorithm from every source independently by a reduction to *dynamic* data structuring problems, e.g., as observed by Chan and Efrat [CE01]. Actually, the reduction is much simplified in the unweighted, undirected setting. However, dynamic data structures for geometric intersection or range searching usually are more complicated and have slower query times than their static counterparts, sometimes by multiple logarithmic factors. For example, the arbitrary disk case employs dynamic data structures for additively weighted nearest-neighbor search. Thus, solving the problem that way takes nearly $O(n^2 \log^{12} n)$ time [KMR 17], while our approach requires only $O(n^2 \log n)$ time.

The second general approach employs *biclique covers* [FM95, AAAS94] to sparsify the intersection graph and then applies an SSSP algorithm from each vertex. However, biclique covers are related to static, offline intersection searching (e.g., as noted in [Cha10b]), which is generally harder than intersection detection. For example, for d-dimensional boxes, the sparsified graph has $O(n \log^d n)$ edges, leading to an $O(n^2 \log^d n)$-time algorithm, but our solution requires $O(n^2 \log^{d-1.5} n)$ time. For arbitrary disks, the complexity of the biclique covers is even worse ($O(n^{3/2})$ [APS93]), leading to an $O(n^{5/2})$-time algorithm, which is much slower than our $O(n^2 \log n)$ solution.

We also solve SSSP in unweighted intersection graphs of n axis-aligned line segments in $O(n \log n)$ time by a reduction to decremental orthogonal point location. Our approach is similar to that of Chan and Efrat [CE01], but simpler because we study the unweighted version of the problem.

4.1 Reducing SSSP to decremental intersection detection

We now show how to reduce SSSP in unweighted, undirected geometric intersection graphs to decremental intersection detection. We give a BFS-like approach, similar to that in Section 2.3. See Algorithm 4.1 for the pseudocode. Specifically, we perform n 1 steps, where in the beginning of each step ℓ, we assume that we have properly processed each $t \in S$ at distance at most ℓ 1 from s. We call the rest of the objects *undiscovered* and those at distance exactly ℓ 1 from s *frontier*. As in classical BFS, we want to find every undiscovered object that shares an edge with a frontier object, but inspecting all edges incident to the latter ones would require quadratic total time. However, in geometric intersection graphs, an edge between two objects exists if and only if they intersect each other, so it suffices to find for each undiscovered object one, if any, frontier object that intersects it. To do so, we would like to solve the following subproblem.

Subproblem 4.1. (Intersection Detection) *Preprocess a set of input objects into a data structure, such that we can quickly decide if a given query object intersects any input object, and, if it does, report any such object.*

An object is undiscovered only until we find a frontier object that intersects it, so we can employ a decremental intersection detection data structure for the former objects and query it in each step of the algorithm with the latter. Whenever we detect an intersection, we properly set the distance and predecessor of the relevant undiscovered object and delete it from the data structure.

Theorem 4.1. (SSSP in unweighted, undirected geometric intersection graphs) *We can solve SSSP in an unweighted, undirected geometric intersection graph of n vertices in $O(DI(n, n) + n)$ time, where $DI(n, m)$ is the required time to construct a decremental intersection detection data structure of n objects and perform n deletions and m queries.*

Proof. In unweighted, undirected geometric intersection graphs, there is an edge between two objects if and only if they intersect, so the correctness of our algorithm follows from that of BFS. The running time is dominated by the time required to construct, update, and query the decremental intersection detection data structure of the undiscovered objects. Initially, each object except the source is undiscovered, and once we delete it from the data structure, we never reinsert it. Thus, the total number of deletions is $O(n)$. There are two cases for a query with a frontier object z. If an undiscovered object t that intersects z is returned, we delete t from the data structure and never reinsert it, so this case happens once for each $t \in S$. If nothing is returned, z does not perform any other queries in that step and, since z is a frontier point only once, this case also happens once $@z \in S$. Thus, the total number of queries is $O(n)$, implying that our algorithm takes $O(DI(n, n) + n)$ time. □

Algorithm 4.1: GeoSSSP($S; s$)

```
1  distrs; ss    0
2  distrs; ts    8, @t P S    tsu
3  predrs; ts    NULL, @t P S
4  build a decremental intersection detection data structure for S    tsu
5  F    tsu
6  for `    1 to n    1 do
7      F^1    ?
8      for each z P F do
9          while true do
10             query the data structure with z and let t be the answer
11             if t not NULL then
12                 distrs; ts    `
13                 predrs; ts    z
14                 delete t from the data structure
15                 F^1    F^1 Y ttu
16             else
17                 break
18     F    F^1
19 return distrs; s and predrs; s
```

Application to axis-aligned line segments. For intersection graphs of axis-aligned line segments, we need a decremental intersection data structure for horizontal input segments and vertical query segments. Vertical input segments and horizontal query segments can be handled with a symmetrical structure. We can use the data structure of either Giyora and Kaplan [GK09, Theorem 5.3] or of Blelloch [Ble08, Theorem 6.1], both of which support vertical ray shooting queries and insertions and deletions of horizontal segments in logarithmic time. Thus $DI\,pn; nq \quad O\,pn \log nq$.

New Result 4. (SSSP in unweighted intersection graphs of axis-aligned line segments) *We can solve SSSP in an unweighted intersection graph of n axis-aligned line segments in $O\,pn \log nq$ time.*

The above theorem can easily be extended to any set of line segments with a constant number of different orientations by constructing one instance of the data structure of Giyora and Kaplan or of Blelloch per orientation.

Remark: Recently, Chan and Tsakalidis [CT18] improved the results of Giyora and Kaplan and of Blelloch to support vertical ray shooting queries in $O\left(\frac{\log n}{\log\log n}\right)$ time and updates in near $O\left(\sqrt{\log n}\right)$ time, assuming that the coordinates are polynomially-bounded integers. Consequently, the time bound in New Result 4 can be improved to $O\left(n \frac{\log n}{\log\log n}\right)$.

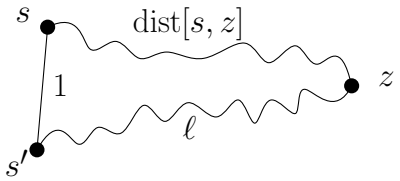
4.2 Reducing APSP to static, offline intersection detection

Here, we reduce APSP in unweighted, undirected geometric intersection graphs to static, offline intersection detection. We first build an arbitrary spanning tree T_0 of G, root it at an arbitrary object $s_0 \in S$, and compute the shortest-path tree of s_0. Then, we visit each object s of T_0 in a pre-order manner and compute its shortest-path tree by using that of s^{-1} as a guide, where s^{-1} is the parent of s in T_0. See Algorithm 4.2 for the pseudocode.

Algorithm 4.2: GeoAPSP(S)

1 build the unweighted, undirected geometric intersection graph G of S
2 compute any spanning tree T_0 of G and root it at any $s_0 \in S$
3 compute $T[s_0]$
4 **for** *each* $s \in S \setminus \{s_0\}$ *following a pre-order traversal of* T_0 **do**
5 compute $T[s]$ by using $T[s^{-1}]$ as a guide, where s^{-1} is the parent of s in T_0
6 **return** T

It remains to describe how to compute the shortest-path tree from an object $s \in S$, given the shortest-path tree from its parent s^{-1} in T_0 (Line 5 in Algorithm 4.2). We first notice that from the triangle inequality and from $dist(s, s^{-1}) \leq 1$, if $dist(z, s^{-1}) = \ell$ for some $z \in S$, then $\ell - 1 \leq dist(z, s) \leq \ell + 1$. That is, we already have an additive-1 approximation of the distances from s; see Figure 4.1. To compute the exact distances from s, we devise a BFS-like algorithm of n steps, similarly to that of the previous section, but now in each step ℓ, we do not have to consider all undiscovered objects. Specifically, the additive-1 approximation of the distances from s implies that the only undiscovered objects that can be at distance ℓ from it are the ones at distance $\ell - 1$, ℓ, or $\ell + 1$ from s^{-1}. We call these *candidate* objects. Thus, we need to find for each candidate whether there is any frontier object that intersects it, i.e., solve the intersection detection problem (Subproblem 4.1). However, contrary to the previous section, the input (frontier objects) here is *static*, and the queries (candidate objects) are *offline*, i.e., are all given in advance. The pseudocode is presented in Algorithm 4.3.



$$\ell - 1 \leq \text{dist}[s, z] \leq \ell + 1$$

Figure 4.1: additive-1 approximation.

Algorithm 4.3: GeoGuideSSSP$(S; s; T(s^1))$

```
1  dist(s) ← 0
2  dist(z) ← ∞ ∀ z ∈ S − {s}
3  pred(z) ← NULL ∀ z ∈ S
4  for ℓ ← 0 to n − 1 do
5      A_ℓ ← {z | dist(z) = ℓ}
6  for ℓ ← 1 to n − 1 do
7      F ← {z ∈ S | dist(z) = ℓ − 1}
8      C ← A_{ℓ−1} ∪ A_ℓ ∪ A_{ℓ+1}
9      build a static, offline intersection detection data structure for F and C
10     for t ∈ C do
11         if dist(t) = ∞ then
12             query the data structure for t
13             let w be the answer
14             if w not NULL then
15                 dist(t) ← ℓ
16                 pred(t) ← w
17 return T(s)
```

Theorem 4.2. (Computing shortest-path trees with 1-additive approximation) *Given a set S of n objects and the shortest-path tree of $s^1 \in S$ in the unweighted, undirected intersection graph of S, we can compute the shortest-path tree from an neighbor $s \in S$ of s^1 in $O(SI(n; n))$ time, where $SI(n; m)$ is the time to construct a static, offline intersection detection data structure for n objects and query it m times. We assume that $SI(n_1; m_1) + SI(n_2; m_2) \geq SI(n_1 + n_2; m_1 + m_2)$.*

Proof. As discussed above, the 1 approximation of the distances from s implies that the candidate objects are the only undiscovered objects that need to be considered in each step. That, along with the argument in the proof of Theorem 19, yields the correctness of our algorithm. Let n_ℓ (respectively m_ℓ) be the number of frontier (respectively candidate) objects in the step ℓ of our algorithm. An object is a frontier object exactly once and a candidate object at most thrice, i.e., $\sum_{\ell=1}^{n-1} n_\ell \leq n$ and $\sum_{\ell=1}^{n-1} m_\ell \leq 3n$. Thus, the time to compute the shortest-path tree from s is $O\left(\sum_{\ell=1}^{n-1} SI(n_\ell; m_\ell)\right) = O(SI(n; n))$. □

Theorem 4.3. (APSP in unweighted, undirected geometric intersection graphs) *We can solve APSP in an unweighted geometric intersection graph of n objects in $O(n^2 + nSI(n; n))$ time, where $SI(\cdot; \cdot)$ is as in Theorem 3.8.*

Proof. In Lines 1–3 of Algorithm 4.2, we can build G, find a spanning tree T_0, and compute the shortest-path tree from s_0 naively in $O(n^2)$ total time. By Theorem 3.8, each of the $n - 1$ iterations of Line 3 of Algorithm 4.2 takes $O(SI(n; n))$ time . □

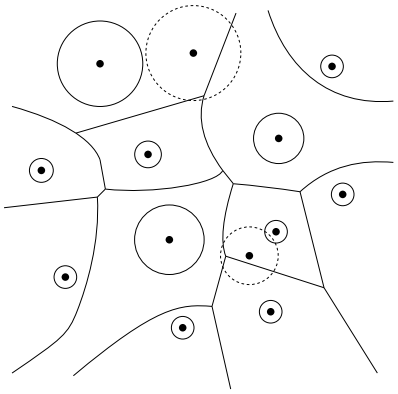


Figure 4.2: A set of input disks (solid), their additively weighted Voronoi diagram (not an accurate one), and two query disks (dashed).

4.2.1 Applications

In this section, we use known results to provide static, offline intersection detection data structures for a family of geometric objects and then apply Theorem 4.3 to solve APSP in the corresponding intersection graphs. Let $SI(\cdot;\cdot)$ be as in Theorem 3.8.

Arbitrary disks in the plane. Given a set of n disks, we create an additively weighted Voronoi diagram and a point location data structure for the cells of that diagram. The sites are the disk centers, and the weight of each site is equal to the radius of the corresponding disk, i.e., the distance between a site p corresponding to a disk of radius r_p and a point q is defined as $d(p;q) = \|p - q\| - r_p$. To detect an intersection of a query disk with an input disk of radius r_q that is centered at q, we can find the input point p that minimizes $d(p;q)$ and check whether $d(p;q) \leq r_q$. Employing the additively-weighted Voronoi diagram algorithm of Fortune [For87] together with the point location data structure of Edelsbrunner et al. [EGS86], we have $SI(n;n) = O(n \log n)$. See Figure 4.2.

Theorem 4.4. *We can solve APSP in an unweighted intersection graph of n disks in $O(n^2 \log n)$ time.*

Axis-aligned line segments in the plane. To construct a static, offline intersection data structure for n axis-aligned line segments we can without loss of generality consider only horizontal input segments and only vertical query segments. We build the vertical decomposition of the former and store it in a point location data structure. Then, given a vertical query segment, we perform a point location query with its endpoints. If they lie in the same cell, there is no intersection. Otherwise, we pick the cell that contains the bottom endpoint of the query segment and report the input segment that bounds its upper side. See Figure 4.3.

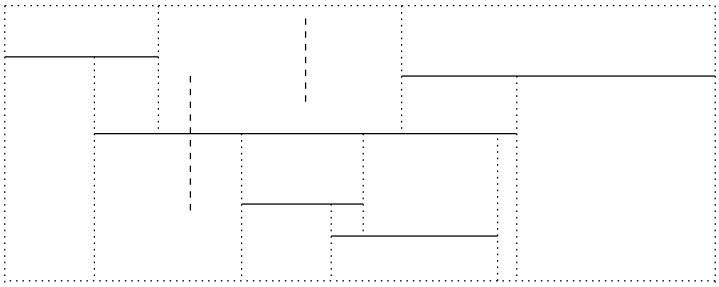


Figure 4.3: A set of horizontal input segments (solid), its vertical decomposition (dotted), and two query vertical segments (dashed).

The point location data structure we employ is the static orthogonal point location data structure of Chan [Cha13, Theorem 2.1], requiring $O(n \log\log U)$ preprocessing and $O(\log\log U)$ query time for n input segments, where U is the universe size. By presorting all coordinates in $O(n \log n)$ time and replacing each coordinate with its rank, we ensure that $U \leq n$. Hence, $SI(n; n) \leq O(n \log\log n)$.

Theorem 4.5. *We can solve APSP in an unweighted intersection graph of n axis-aligned line segments in $O(n^2 \log\log n)$ time.*

The result can be easily be extended to any set of line segments with a constant number of different orientations by constructing one instance of Chan's data structure per orientation.

Arbitrary line segments. Chazelle's $O(n^{4/3} \log^{1/3} n)$-time algorithm [Cha93, Theorem 4.4] for counting the number of intersections among n line segments can be extended to count the number of intersections between n red (input) line segments and n blue (offline query) line segments. In fact, the algorithm can decide whether each blue segment intersects any red segment, and if so, report one such red segment. Thus, $SI(n; n) \leq O(n^{4/3} \log^{1/3} n)$.

Theorem 4.6. *We can solve APSP in an unweighted intersection graph of n arbitrary line segments in $O(n^{7/3} \log^{1/3} n)$ time.*

Axis-aligned boxes in d dimensions. Offline rectangle intersection counting in n axis-aligned rectangles in the plane is known to be reducible [EO82] to offline orthogonal range counting, for which Chan and Pătraşcu [CP10, Corollary 2.3] have given an $O(n \sqrt{\log n})$-time algorithm, assuming that all coordinates have been presorted. Thus, we can decide for each query box whether it intersects any input box. In Section 4.3, we adapt the technique of Chan and Pătraşcu to construct a data structure that can also report such an input box if it exists. Hence, after presorting in $O(n \log n)$ time, we have that $SI(n; n) \leq O(n \sqrt{\log n})$.

For axis-aligned boxes in $d \geq 3$ dimensions, we use standard range trees [EM81] with the above planar base case to obtain $SI(n, n) = O(n \log^{d-1.5} n)$.

Theorem 4.7. *We can solve APSP in an unweighted intersection graph of n d-dimensional axis-aligned boxes in $O(n^2 \log^{d-1.5} n)$ time, for $d \geq 3$.*

Axis-aligned unit hypercubes in d dimensions. We can construct more efficient static, offline data structures when the n boxes are unit hypercubes. Specifically, we first build a uniform grid with unit side length and then solve the problem inside each grid cell separately. Each input or query unit hypercube participates in at most a constant (2^d) number of grid cells, and inside each of them, it is effectively unbounded along d sides. We assume without loss of generality that each input box is of the form $(-\infty, a_1] \times \cdots \times (-\infty, a_d]$ and that each query box is of the form $[b_1, +\infty) \times \cdots \times [b_d, +\infty)$. Thus, the problem reduces to offline *dominance* detection: decide for each query point $(b_1, \ldots, b_d)$ whether it is dominated by some input point $(a_1, \ldots, a_d)$, and, if yes report one such input point.

For $d = 3$, the algorithm of Gupta et al. [GJSD97, Theorem 3.1] answers n offline dominance reporting queries in $O((n + K) \log \log U)$ time, where n is the input size, and K is the total output size, assuming that all coordinates are integers bounded by U. Thus, n offline dominance detection queries can be answered in $O(n \log \log U)$ time. By presorting all coordinates in $O(n \log n)$ time and replacing each with its rank, we ensure that $U = n$. Hence, $SI(n, n) = O(n \log \log n)$. For $d \geq 4$, Afshani et al. [ACT14, Remark in Section 5] (following Chan et al. [CLP11]) has given a deterministic algorithm to answer n offline dominance reporting queries in $O(n \log^{d-3} n + K)$ time, where n is the input size, and K is the total output size. It can be checked that n offline dominance detection queries can be answered in $O(n \log^{d-3} n)$ time.

Theorem 4.8. *We can solve APSP in an unweighted intersection graph of n d-dimensional axis-aligned unit hypercubes in $O(n^2 \log \log n)$ time for $d = 3$ and in $O(n^2 \log^{d-3} n)$ time for $d \geq 4$.*

Fat triangles in the plane. Given n *fat* triangles (i.e., triangles that have bounded inradius-to-circumradius[1] ratios) of roughly equal sizes, Katz [Kat97, Theorem 4.1 (i)] has shown how to construct an intersection reporting data structure of $O(n \log^4 n)$ preprocessing and $O(\log^3 n + K \log^2 n)$ query time, where n is the input size, and K is the total output size (here $K \geq 1$). Hence, $SI(n, n) = O(n \log^4 n)$.

Theorem 4.9. *We can solve APSP in an unweighted intersection graph of n fat triangles of roughly equal sizes in $O(n^2 \log^4 n)$ time.*

[1] The inradius of a triangle is the radius of the biggest circle that can be drawn inside it. The circumradius of a triangle is the radius of the the circle that passes through its three vertices.

4.3 Static, offline rectangle intersection detection

We now show how to construct a data structure for the static, offline rectangle intersection detection problem: given a set of n input and query rectangles, find for each query rectangle one, if any, input rectangle that intersects it. We assumed the existence of such a structure to obtain the APSP algorithm of Theorem 4.7 for unweighted, undirected graphs of d-dimensional boxes. As depicted in Figure 4.4, we can easily reduce that problem [EM81] to the following subproblems:

(i) Axis-aligned line segment intersection detection: finding some input horizontal/vertical segment that intersects a query vertical/horizontal segment.

(ii) Orthogonal range detection: finding some input point inside a query rectangle.

(iii) *Rectangle stabbing* detection: finding some input rectangle that contains a query point).

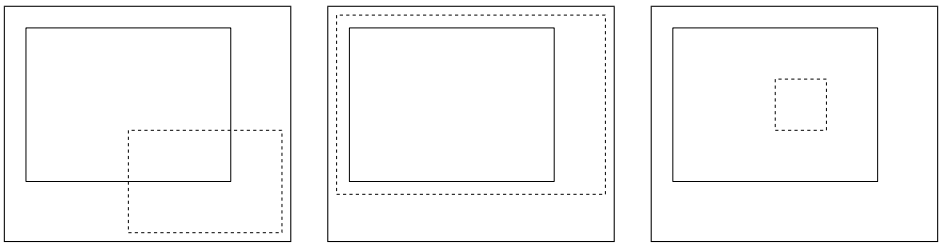


Figure 4.4: Axis-aligned line segment intersection detection, orthogonal range detection, and rectangle stabbing detection. The input rectangles are solid, while the query rectangles are dashed.

In Section 4.2.1, we showed how to answer n offline queries of type (i) in $O(n \log\log U)$ time, assuming that the coordinates of the vertices of the rectangles are integers bounded by U. Also, Babenko et al. [BGKS15] have already described how to adapt Chan and Pătraşcu's technique [CP10] to answer n offline queries of type (ii) in $O(n\sqrt{\log n})$ time. Babenko et al. actually solved the *range successor* problem, which is equivalent to finding the lowest point in a 3-sided query rectangle unbounded from above—it is easy to see that 4-sided orthogonal range detection reduces to this problem. We now describe how to adapt Chan and Pătraşcu's technique to answer n offline queries of type (iii) in $O(n\sqrt{\log n})$ time. Chan et al. [CLP11] noted a similar result but only for the case of *disjoint* rectangles.

Theorem 4.10. (Rectangle Stabbing Detection) *Given n input axis-aligned rectangles and n planar query points with presorted coordinates, we can report for each query point, an input rectangle, if any, that contains it in $O(n\sqrt{\log n})$ time.*

Proof. We use words of $w = \log n$ bits, where $0 < \varepsilon < 1$ is a constant. Also, we assume, without loss of generality, that all y-coordinates are distinct.

Special case: all x-coordinates are small integers bounded by s. We employ a divide-and-conquer scheme that resembles a binary interval tree in the x-axis and bit-packing techniques. The input of our algorithm is the list of the x-coordinates of the vertices of the input rectangles and of the query points, presorted by y. Thus, it can be packed with $O(n \log s/w)$ words. The output is represented as the list of the minimum and maximum x-coordinates of a rectangle, if any, that contains each query point in bottom-to-top order. Hence, it can also be packed with $O(n \log s/w)$ words.

Let R be the set of input rectangles and Q that of query points. First, we find the vertical line $x = m$ that divides the x-universe into two halves of length $s/2$. Let R_m be the subset of rectangles that intersect that line. The union of R_m is a y-monotone polygon or multiple such polygons, and we can find it by computing the left/right envelope of the vertical line segments of the input rectangles. Eppstein and Muthukrishnan [EM01] have shown how to do that in $O(|R_m|)$ time, assuming that the coordinates have been pre-sorted in x and y. In our case, the latter has already been done, and to obtain the former we employ counting sort in $O(|R_m| + s)$ time. Then, we solve the problem for R_m and Q with a bottom-to-top scan, using $O(n \log s/w)$ additional word operations.

Next, let R_ℓ (respectively R_r) be the subset of rectangles completely to the left (respectively right) of $x = m$ and Q_ℓ (respectively Q_r) the subset of query points to the left (respectively right) of $y = m$. We recursively solve the subproblem for R_ℓ and Q_ℓ and the subproblem for R_r and Q_r. The input to either subproblem can be formed by a linear scan using $O(n \log s/w)$ word operations, and the output can be merged by another linear scan using $O(n \log s/w)$ word operations.

Excluding the cost of computing the unions of the R_m's, the total running time is $O(n \log^2 s/w + s \log s)$ since there are $O(\log s)$ levels of recursion. Each rectangle lies in exactly one subset R_m over the entire recursion tree, so the total cost of computing the unions of the R_m's is $O(n)$.

One remaining issue is that the output only records the x-coordinates of the reported rectangles. To retrieve the y-coordinates, we first partition the original set of input rectangles into $O(s^2)$ classes with common minimum and maximum x-coordinates. For each query point, we have identified one class which contains an answer. For each class C, we can gather its input rectangles R_C and query points Q_C, both pre-sorted by y, and answer these queries. This is a 1-dimensional problem in y (finding an input interval containing each query point), which is a special case of the above-mentioned envelope problem and can be solved in $O(|R_C| + |Q_C|)$ time, thus the total time over all classes C is linear. We conclude that the special case case can be solved in $O(n + n \log^2 s/w)$ time, assuming that $n \geq s^2$.

General case. We again use a divide-and-conquer approach, but this time in a degree-s-segment-tree manner. We use $s - 1$ vertical lines to divide the plane into s slabs each with $O(n/s)$ rectangle vertices and query points. Each rectangle can be divided into at most three parts, where the left (respectively right) part is contained in one of the s slabs, and

the middle part has x-coordinates aligned with the dividing vertical lines. For all middle parts, we can round the x-coordinates of the query points to align with the dividing lines and apply the algorithm for the above special case in $O(n \log^2 s)$ time. For the left and right parts, if $n \geq s^2$, we recursively solve the subproblems inside the s slabs. We can combine the answers in linear time.

Each rectangle and each query point participates in $O(\log_s n)$ recursive calls. The total time is thus $O(n \log^2 s + n \log n (\log s))$. Setting $\log s = \sqrt{w}$ yields $O(n \log n (\log n) / \sqrt{w})$, assuming that $n \geq 2^{\sqrt{w}}$. Since $w = \log n$, where $0 < < 1$ is a constant, we can perform each of the above word operations in constant time with table lookup, after an initial precomputation in $o(n)$ time (see Section 2.1). We can also handle the base case, $n = s^2$, in linear total time with brute force. □

As explained in the beginning of this section, we can use our static, offline rectangle stabbing detection data structure of Theorem 4.10 to construct a data structure for static, offline rectangle intersection detection.

Corollary 4.11. (Static, offline rectangle intersection detection) *Given n input and n query axis-aligned rectangles with presorted coordinates, we can report an input rectangle for each query rectangle (if it exists) that intersects it in $O(n \sqrt{\log n})$ time.*

Chapter 5

Approximate diameter and distance oracles in planar graphs

In this chapter, we study approximate shortest-path problems in planar graphs. Namely, we present an $O(n \log n \log\log n \cdot (1/\varepsilon)^{5})$-time algorithm that computes a $(1+\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph of n vertices. Therefore, we improve upon the $O(n \cdot (1/\varepsilon)^{4} \log^{4} n \cdot 2^{O(1/\varepsilon)})$-time algorithm of Weimann and Yuster [WY16] in two regards. First, we eliminate the exponential dependency on $1/\varepsilon$ by adapting and specializing Cabello's recent Voronoi-diagram-based technique [Cab17a] for approximation purposes. Second, we shave off two logarithmic factors by choosing a better sequence of error parameters in the recursion and by employing the multiple shortest paths data structure of Klein [Kle05].

Moreover, using similar techniques we obtain a variant of Gu and Xu's $(1+\varepsilon)$-approximate distance oracle [ISAAC 2015] with polynomial dependency on $1/\varepsilon$ in the preprocessing time and space and $O(\log(1/\varepsilon))$ query time.

The results of this chapter have been presented in ESA 2017 [CS17b].

Definitions. Let $G = (V, E)$ be a non-negatively-weighted, undirected planar graph, i.e., a graph that can be drawn in the plane, such that edges intersect only at their endpoints. We also refer to V and E as $V(G)$ and $E(G)$ respectively. Let G^* be the dual of G, i.e., the graph whose vertices correspond to faces of G (and vice versa), and there is an edge between two vertices of G^* if and only if the corresponding faces in G share a common edge. We assume that G and any graph under discussion in this chapter is triangulated (we can triangulate naively in linear time) and comes with a fixed embedding (we can find such an embedding in linear time [HT74]).

For any $u, v \in V$, we denote a u-to-v shortest path in G by $\pi_G(u, v)$ and its length by $dist_G(u, v)$. We also refer to $dist_G(u, v)$ as shortest-path distance or simply distance. Let $pred_G(u, v)$ be v's predecessor on $\pi_G(u, v)$. The shortest-path tree of each $u \in V$ is a

spanning tree of G rooted at u, such that the u-to-v shortest-path distance for each $v \in V$ in that tree corresponds to $dist_G(u,v)$. We are interested in the following shortest-path problems in G.

- Computing a $(1+\varepsilon)$-approximation of various parameters of G, such as the *diameter* (i.e., $\max_{u,v \in V} dist_G(u,v)$), the *radius* (i.e., $\min_{u \in V} \max_{v \in V} dist_G(u,v)$), the *Wiener index* (i.e., $\sum_{u,v \in V} dist_G(u,v)$), the *eccentricity* of each vertex $u \in V$ (i.e., $\max_{v \in V} dist_G(u,v)$), et cetera.
- Constructing *approximate distance oracles*, i.e., data structures that support the following query: given any $s,t \in V$, compute a value $\hat{d}$ with $dist_G(s,t) \le \hat{d} \le (1+\varepsilon)dist_G(s,t)$ and the predecessor of t in an s-to-t path of length $\hat{d}$.

Background and overview of techniques.

Diameter. Since Frederickson [Fre87] presented an $O(n^2)$-time exact diameter algorithm (by solving APSP) for non-negatively-weighted planar graphs of n vertices, a natural question arose as to whether the diameter can be computed in truly subquadratic time. Eppstein [Epp99] gave a partial answer for the unweighted case by proving that if the diameter is constant, it can be found in linear time. Later, Chan [Cha12] and Wulff-Nilsen [WN10] developed two slightly-subquadratic-time solutions (for arbitrary diameter), both requiring $O\left(n^2 \frac{\log\log n}{\log n}\right)$ time in unweighted graphs. The algorithm of Wulff-Nilsen also works for the weighted case but in $O\left(n^2 \frac{(\log\log n)^4}{\log n}\right)$ time.

In 2017, Cabello [Cab17a] (full paper in [Cab17b]) made a breakthrough by giving an $\tilde{O}(n^{11/6})$-expected-time algorithm, where $\tilde{O}(f(n))$ denotes $O(f(n)\log^{O(1)} n)$. His techniques are as interesting as the result itself, for they involved a seemingly alien concept to planar graphs, *Voronoi diagrams*, originating from computational geometry. Gawrychowski et al. [GKM+18], again using Voronoi diagrams, derandomized Cabello's algorithm and improved its running time to $\tilde{O}(n^{5/3})$.

In the approximate setting, faster algorithms are known for undirected planar graphs. The linear-time SSSP algorithm of Henzinger et al. [HKRS97] can trivially compute a 2-approximation, while Berman et al. [BK07] developed a (3/2)-approximation algorithm of $O(n^{3/2})$ time. Weimann and Yuster [WY16], in a breakthrough, presented the first $(1+\varepsilon)$-approximation algorithm, running in $\tilde{O}(n(1/\varepsilon)^4 \log^4 n \cdot 2^{O(1/\varepsilon)})$ time. However, that does not settle the problem because of the *exponential* dependency on $1/\varepsilon$ and of the multiple (four) $\log n$ factors.

We show that Cabello's technique of employing Voronoi diagrams in planar graphs can be combined nicely with Weimann and Yuster's recursive scheme to eliminate the $2^{O(1/\varepsilon)}$ factor from the running time of the latter. Compared with Cabello's approach,

which had to deal with the general case of site weights being real numbers, our version of Voronoi diagrams is much simplified because in the approximate setting we can map the site weights to small integers. We also eliminate two of the four logarithmic factors, by using a better sequence of error parameters in the recursion and by employing the multiple shortest paths data structure of Klein [Kle05].

Specifically, we describe in Section 5.1 an efficient $(1+\varepsilon)$-approximate farthest neighbor data structure in planar graphs by employing Voronoi diagrams. Then, we describe in Section 5.2 how to use that data structure in the framework of Weimann and Yuster to compute a $(1+\varepsilon)$-approximation of the length of the longest shortest path distance of every graph encountered during the recursion.

Distance oracles. Thorup [Tho04a], in a seminal paper, gave a $(1+\varepsilon)$-approximate distance oracle for non-negatively-weighted, undirected planar graphs of n vertices, requiring $O((1/\varepsilon^2)n\log^3 n)$ preprocessing time, $O((1/\varepsilon)n\log n)$ space, and $O((1/\varepsilon)\log n)$ query time. That oracle was later simplified by Klein [Kle02]. Kawarabayashi et al. [KKS11] constructed an oracle of linear space but $O((1/\varepsilon^2)\log^2 n)$ query time, and then Kawarabayashi et al. [KST13] improved the dependency on $1/\varepsilon$ of the space-query-time product from $1/\varepsilon^2$ to $1/\varepsilon$. In the Word RAM, Gu and Xu [GX15] combined the techniques of the above results with those of Weimann and Yuster [WY16] for the approximate diameter problem to obtain the first distance oracle with constant query time (independent of both n and ε). However, the preprocessing time and space of their data structure have exponential dependency on $1/\varepsilon$ (they are $O(n\log n+(1/\varepsilon^2)\log^3 n)2^{O((1/\varepsilon)\log(1/\varepsilon))}$ and $O(n\log n+(1/\varepsilon)n\log n)2^{O((1/\varepsilon)\log(1/\varepsilon))}$ respectively).

We employ techniques similar to those of our diameter algorithm to develop in the Word RAM the first $(1+\varepsilon)$-approximate distance oracle with $o((1/\varepsilon)\log n)$ query time and $O((1/\varepsilon)^{O((1/\varepsilon)\log(1/\varepsilon))}n\log^{O((1/\varepsilon)\log(1/\varepsilon))}n)$ preprocessing time and space. Specifically, the preprocessing time and space of our oracle are $O(n\log^2 n\,\mathrm{plog}\,n+(1/\varepsilon)^5(1/\varepsilon))$ and $O((1/\varepsilon)n\log^2 n)$ respectively, and the query time is $O(\mathrm{plog}((1/\varepsilon)(1/\varepsilon)))$. Although we slightly increase the query time of the oracle of Gu and Xu (from $O((1/\varepsilon)(1/\varepsilon))$ to $O(\mathrm{plog}((1/\varepsilon)(1/\varepsilon)))$), we significantly improve its preprocessing time and space by eliminating the exponential dependency on $1/\varepsilon$.

In Section 5.3.1 we construct an oracle of additive stretch, and then in Section 5.3.2 we show how to use that oracle along with existing techniques, namely scaling [KST13, GX15], to obtain our $(1+\varepsilon)$-approximate distance oracles.

5.1 A farthest-neighbor data structure

Here, we want to construct a data structure for the following *farthest-neighbor* problem in planar graphs, which is crucial in obtaining our diameter algorithm in Section 5.2. Let $[W]=\{0,1,\ldots,W-1\}$ for an integer $W\geq 0$.

Problem 5.1. (Farthest neighbor) *Let $H = (V, E)$ be a triangulated planar graph of n vertices with a fixed embedding. Also, let X be a set of b vertices on the boundary of its outer face, and let U be a subset of V. Finally, let H' be the graph that is obtained by adding to H a vertex z_0 and an edge of unspecified weight from z_0 to each $x \in X$.*

Construct a data structure that supports the following kind of queries for a fixed integer W: given a weight $\ell(z_0, x) \in [W]$ for each edge (z_0, x), where $x \in X$, find the distance to the farthest neighbor of z_0 in H' among the vertices of U, i.e., compute $\max_{u \in U} dist_{H'}(z_0, u)$.

Cabello [Cab17b, Theorem 21] employed Voronoi diagrams in planar graphs to develop a farthest-neighbor data structure with $\mathcal{O}(n^2 b^3 + b^4)$ preprocessing time and $\mathcal{O}(b \log b)$ expected query time, under a non-degeneracy assumption. His result works for the more general version of Problem 5.1 where the weight $\ell(\cdot, \cdot)$ of each edge (z_0, x) $(x \in X)$ is a real number. We show that when these weights are instead small integers, we can employ Voronoi diagrams in a simpler way to construct a farthest-neighbor data structure in time only near linear in n. Moreover, our query time does not have any $\log n$ factors.

5.1.1 Defining Voronoi diagrams in planar graphs

The concepts of standard geometric Voronoi diagrams can easily be extended to the planar graph $H = (V, E)$ from the setting of Problem 5.1. Each *site* s is a pair (v_s, w_s), where v_s is its placement (i.e., a vertex of H), and w_s is its weight. Given a set of sites S, the *graphic Voronoi region* of $s \in S$ in H is defined as $\mathrm{VR}(s, S) = \{u \in V \mid dist_H(v_s, u) + w_s \leq dist_H(v_t, u) + w_t, \forall t \in S\}$, i.e., as the set of all vertices closer to s than to any other site under the weighted metric. Then, the (additively weighted) *graphic Voronoi diagram* $\mathrm{VD}(S)$ of S in H is simply the collection of $\mathrm{VR}(s, S)$ over all $s \in S$. See Figure 5.1(a)-(b). Since we discuss Voronoi diagrams only in H, we drop the subscripts and refer to $dist(\cdot, \cdot)$, $\mathrm{VR}(\cdot, \cdot)$, and $\mathrm{VD}(\cdot)$ from now on.

Henceforth, we assume that S is a set of b sites, each placed at a vertex on the boundary of the infinite face of H. We assume that S is *generic*, i.e., for each $s, t \in S$ and $u \in V$, we have $dist(u, v_s) + w_s \neq dist(u, v_t) + w_t$. We also assume that S is *non-redundant*[1], i.e., each region of the graphic Voronoi diagram of S is non-empty. We ensure that every Voronoi diagram under discussion later is associated with a generic and non-redundant set of sites. We define the *bisector* $\mathrm{bis}(s, t)$ of every $s, t \in S$ to be the set of the duals of the edges in $\{(u, v) \in E \mid dist(u, v_s) + w_s \neq dist(u, v_t) + w_t$ and $dist(v, v_t) + w_t \neq dist(v, v_s) + w_s\}$. In other words, $\mathrm{bis}(s, t)$ is composed of the duals of the edges whose endpoints are not both closer to the same site. From [Cab17b, Lemma 10], each such bisector is a simple cycle in H. See Figure 5.1(c).

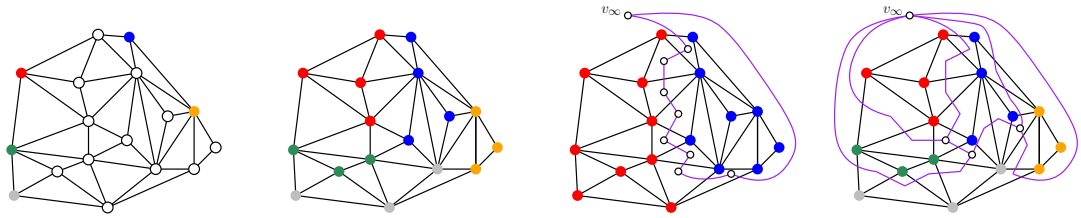
As explained in Section 2.6, Klein [Kle89] introduced the framework of *abstract Voronoi diagrams* to unify the treatment of various Voronoi diagrams in the plane. Cabello

[1] Cabello [Cab17b] used the term *independent* in his paper.

defined a system of *abstract bisectors* as $\mathrm{bis}(s,t) = \bigcap_{v \in V(R(s,t,u))} \overline{v}$ $(s,t \in S, s \neq t)$,

where v is the dual face of $v \in V$, $\overline{A}$ is the closure of $A \subseteq \mathbb{R}^2$, and A is the interior of A. Also, Cabello showed [Cab17b, Lemma 12] that his system of abstract bisectors fulfills Klein's *admissibility* axioms, as long as S is generic and non-redundant. Thus, as mentioned in Section 2.6, the graphic Voronoi diagram of S can be implicitly represented as a planar graph with a fixed combinatorial embedding of $O(n)$ *Voronoi arcs* and *Voronoi nodes* (an explicit representation would require $O(n^2)$ time). Each such Voronoi arc is a contiguous portion of a bisector (thus, a simple path in H), and each Voronoi node is a vertex of H incident to three Voronoi arcs. Henceforth, every reference to a Voronoi diagram is to the implicit representation of the corresponding graphic Voronoi diagram in H. See Figure 5.1(d).

We can compute the Voronoi diagram of S with the following algorithm of Klein et al. [KMM93, Theorem 1], which is an extension of a standard randomized incremental construction algorithm for Euclidean Voronoi diagrams [CS89, Mul94]. The worst-case running time of that algorithm is quadratic. Also, each Voronoi arc (respectively, node) is represented with a pointer to a Voronoi arc (respectively, node) in the Voronoi diagram of four sites of S [Cab17b, Section 4]. We state the result of Klein et al. tailored for our setting.



(a) (b) (c) (d)

Figure 5.1: (a) Five sites (the coloured vertices) of a planar graph. (b) Their graphic Voronoi diagram. (c) A bisector of the blue and red sites. (d) The Voronoi diagram of the five sites.

Theorem 5.1. (Via abstract Voronoi diagrams) *Let H be a triangulated planar graph with a fixed embedding, and let S be a generic and non-redundant set of b sites placed on vertices on the boundary of its outer face. We can construct the Voronoi diagram of S with an expected number of $O(b \log b)$ or a worst-case number of $O(b^2)$ elementary operations. Here, an elementary operation is the computation of the Voronoi diagram of any four sites of S.*

5.1.2 Constructing Voronoi diagrams in planar graphs

We now show how to perform the elementary operations of the Voronoi diagrams algorithm of Theorem 5.1, i.e., compute the Voronoi diagram of any four sites of a given generic and

non-redundant set $S^1 \subseteq S$. For the rest of this section, we assume that each $s \in S^1$ is assigned a weight drawn from rW. We also assume that without loss of generality all subsets of at most four sites of S are generic and non-redundant (we can easily find those that are not and ignore them).

As Cabello did, we first compute all possible bisectors for each pair of sites of S. Our approach builds upon that of [Cab17b, Lemma 17] and requires only $O(nW)$ time, whereas Cabello's needed $O(n^2)$ time for site weights being real numbers. Also, we can return a pointer to a bisector in $O(1)$ time instead of $O(\log n)$ time.

Lemma 5.2. (Bisectors) *Let H be a triangulated planar graph with a fixed embedding, and let S be a set of b sites placed on vertices on the boundary of its outer face with unspecified weights. Given two sites $s, t \in S$, there are $O(W)$ distinct bisectors in the family of* $\mathrm{bis}(v_s, w_s, v_t, w_t)$ *over all possible weights w_s and w_t drawn from* rW*. Moreover, we can compute all of them in $O(nW)$ total time, such that given weights $w_s, w_t \in \mathrm{rW}$ for any generic and non-redundant set of two sites $s, t \in S$, we can return a pointer to their bisector in $O(1)$ time.*

Proof. Assuming without loss of generality that $w_s \neq w_t$, we can write $\mathrm{bis}(v_s, w_s, v_t, w_t)$ as $\mathrm{bis}(v_s, w, v_t, 0)$, where $w = w_s - w_t \in \mathrm{rW}$. Thus, there are $O(W)$ distinct bisectors in that family. We represent each such bisector with a linked list that contains its edges (which form a cycle) in clockwise order.

In the preprocessing phase, we find the shortest-path trees from v_s and v_t in linear time [HKRS97] and compute the value $d_u = \mathrm{dist}(v_t, u) - \mathrm{dist}(v_s, u)$ for each $u \in V$. For each of the $O(W)$ values of w, we find with a linear scan every edge $uv \in E$, such that $w \neq d_u$ and $w \neq d_v$, and insert its dual to the linked list of $\mathrm{bis}(v_s, w, v_t, 0)$. We might need to rearrange that list in linear time. Finally, we store all these lists in a table, such that given weights $w_s, w_t \in \mathrm{rW}$, where without loss of generality $w_s \neq w_t$, for any two sites $s, t \in S$, we can return a pointer to the linked list of $\mathrm{bis}(v_s, w_s - w_t, v_t, 0)$ in $O(1)$ time. □

Next, we show how to compute all Voronoi diagrams of any three sites of S in $O(nW^2)$ time. We improve upon the approach of Cabello [Cab17b, Lemma 18], requiring $O(n^2)$ time for site weights being real numbers, and we also simplify it, as we employ neither line arrangements nor amortization. We can return a pointer to a Voronoi diagram of any three sites of S in $O(1)$ time instead of $O(\log n)$.

Lemma 5.3. (Voronoi diagrams of 3 sites) *Let H be a triangulated planar graph with a fixed embedding, and let S be a set of b sites placed on vertices on the boundary of its outer face with unspecified weights. Given three sites $s, t, q \in S$, there are $O(W^2)$ distinct Voronoi diagrams in the family* $\mathrm{VD}_{\mathrm{pt}}(v_s, w_s, v_t, w_t, v_q, w_q)$ *over all possible weights w_s, w_t, and w_q drawn from* rW*. Moreover, we can compute all of them in $O(nW^2)$ total time, such that given weights $w_s, w_t, w_q \in \mathrm{rW}$ for any generic and non-redundant set of three sites $s, t, q \in S$, we can return a pointer to their Voronoi diagram in $O(1)$ time.*

Proof. As in the proof of Lemma 5.2, assuming without loss of generality that $w_s; w_t \geq w_q$, we can write $\mathrm{VD}(\mathrm{tp}v_s; w_s\mathrm{q}; \mathrm{p}v_t; w_t\mathrm{q}; \mathrm{p}v_q; w_q\mathrm{quq}$ as $\mathrm{VD}(\mathrm{tp}v_s; w_s^1\mathrm{q}; \mathrm{p}v_t; w_t^1\mathrm{q}; \mathrm{p}v_q; 0\mathrm{quq}$, where $w_s^1 \quad w_s \quad w_q; w_t^1 \quad w_t \quad w_q \in \mathrm{r}W\mathrm{s}$. Hence, that family of Voronoi diagrams has size $O(\mathrm{p}W^2\mathrm{q})$. According to [Cab17b, Lemma 13], a Voronoi diagram of three sites has at most one Voronoi node (besides v_8), which we represent with a pointer. Also, each Voronoi arc is a contiguous portion $e_1; e_2; \cdots; e_k$ for some k, of the bisector of $\mathrm{p}s; t\mathrm{q}$, $\mathrm{p}s; q\mathrm{q}$, or $\mathrm{p}t; q\mathrm{q}$ [Cab17b, Section 5.2]. Thus, we represent that arc with pointers to e_1, to e_k, and to the relevant bisector.

In the preprocessing phase, we invoke Lemma 5.2 to compute and store all bisectors of $\mathrm{p}s; t\mathrm{q}$, $\mathrm{p}s; q\mathrm{q}$, and $\mathrm{p}t; q\mathrm{q}$ in $O(\mathrm{p}nW\mathrm{q})$ time. Then we find the shortest-path trees from v_s, v_t, and v_q in linear time [HKRS97], and we compute for each vertex $u \in V$ the values $\overset{st}{u} \quad dist(v_s; u) \quad dist(v_t; u)$, $\overset{qt}{u} \quad dist(v_q; u) \quad dist(v_t; u)$, and $\overset{sq}{u} \quad dist(v_s; u) \quad dist(v_q; u)$.

For each of the $O(\mathrm{p}W^2\mathrm{q})$ values of w_s^1 and w_t^1, we use the values to find with a linear clockwise scan that starts at v_8 the first and the last edge $\mathrm{p}uv\mathrm{q}$ (i.e., the dual of $uv \in E$) of bis$(s; t)$ such that $u \in \mathrm{VR}(s; \mathrm{t}s; t; q\mathrm{uq}$ and $v \in \mathrm{VR}(t; \mathrm{t}s; t; q\mathrm{u})$. If these edges exist, we properly create the pointers for a Voronoi arc and then repeat with bis$(s; q)$ and bis$(t; q)$. If $\mathrm{VD}(\mathrm{t}s; t; q\mathrm{uq}$ has three Voronoi arcs, we can find and store a pointer to its Voronoi node in $O(1)$ time by determining the vertex of H where these arcs meet. Last, we store a pointer to each computed Voronoi diagram in a two-dimensional table, such that given weights $w_s; w_t; w_q \in \mathrm{r}W\mathrm{s}$, where without loss of generality $w_s; w_t \geq w_q$, we can return pointers to the node and to the arcs of $\mathrm{VD}(\mathrm{tp}v_s; w_s \quad s_q\mathrm{q}; \mathrm{p}v_t; w_t \quad w_q\mathrm{q}; \mathrm{p}v_q; 0\mathrm{quq}$ in $O(1)$ time. □

We now give provide a data structure that given any four sites of S, constructs their Voronoi diagram, thus supporting the elementary operation of the Voronoi diagrams algorithm of Theorem 5.1. The proof is similar to that in [Cab17b, Lemma 19], but using Lemmas 5.2 and 5.3 for bisectors and Voronoi diagrams of three sites respectively, we achieve $O(\mathrm{p}nb^3W^2\mathrm{q})$ preprocessing time instead of $\mathcal{O}(\mathrm{p}n^2b^3\mathrm{q})$. Also, the query time here is $O(1)$ instead of $O(\log n)$. The proof we give below is a paraphrase of that of Cabello, and we include it for the sake of completeness.

Lemma 5.4. (Voronoi diagrams of 4 sites) *Let H be a triangulated planar graph with a fixed embedding, and let S be a set of b sites placed on vertices on the boundary of its outer face with unspecified weights. We can construct in $O(\mathrm{p}nb^3W^2\mathrm{q})$ time a data structure that supports the following kind of queries: given a generic and non-redundant set of four sites of S, whose weights are drawn from $\mathrm{r}W\mathrm{s}$, we can return a pointer to their Voronoi diagram in $O(1)$ time.*

Proof. In the preprocessing phase, we apply the methods of Lemmas 5.2 and 5.3 to compute all distinct bisectors (respectively Voronoi diagrams) of any two (respectively three) sites of S in $O(\mathrm{p}nW^2\mathrm{q})$ time. Let $s; t; q; r$ be four given sites of S with weights drawn from $\mathrm{r}W\mathrm{s}$. We assume without loss of generality that the clockwise order of the four sites on the

boundary of the outer face of H is $s; t; r; q$. For each pair $s; t$ we check in constant time whether bis$(s; t)$ fully participates in VD$(ts; t; q)$ and VD$(ts; t; r)$.

If that is so, then bis$(s; t)$ encloses exactly the vertices of H that are closest to s than to any of the other three sites, i.e., the vertices of VR$(s; ts; t; q; r)$. Thus, VD$(ts; t; q; r)$ is composed of bis$(s; t)$ and VD$(tt; q; r)$, and we can easily generate its Voronoi nodes and arcs. Notice that in this case, the part of the Voronoi diagram that is restricted in the interior faces of H (i.e., the diagram after "deleting v_8 and its adjacent edges) is not connected. See Figure 5.2(a).

Else, no bisector fully bounds a Voronoi region. In other words, the part of the Voronoi diagram that is restricted in the interior faces of H is connected. That implies that VD$(ts; t; q; r)$ has two Voronoi nodes, besides v_8 (remember that H is triangulated). See Figure 5.2(b). There are two cases for these nodes. First, they are the meeting points of each bis$(s; q)$ and of each bis$(r; q)$, respectively (Figure 5.2(c)). Second, they are the meeting points of each bis$(q; q)$ and of each bis$(t; q)$, respectively (Figure 5.2(d)).

We can determine which case we are in by finding whether bis$(s; r)$ participates in VD$(ts; t; q; r)$. Notice that the intersection of VR$(s; ts; r; q)$ and VR$(s; ts; r; t)$ gives VR$(s; ts; r; t; q)$ because these two Voronoi diagrams contain each bisector bis$(s; q)$. Thus, we can find the Voronoi nodes v and v^1 of VD$(ts; r; q)$ and VD$(ts; r; t)$ respectively, and compare the order of v_8, v^1, and v along bis$(s; r)$. That can be done in constant time after linear preprocessing time per bisector. If it is clockwise, we are in the first case (see Figure 5.2(e)); else, we are in the second (see Figure 5.2(f)). After determining the case we are in, we can generate the Voronoi nodes and arcs of VD$(ts; t; q; r)$ by properly using the information of the relevant Voronoi diagrams of triples of sites. □

Combining the above lemma with the Voronoi diagram algorithm of Theorem 5.1, we obtain a data structure that given any generic and non-redundant set $S^1 \subset S$, whose weights are drawn from $[W]$, computes their Voronoi diagram. Its preprocessing time is $O(nb^3W^2)$, while the structure of Cabello required $\Theta(n^2b^3)$ construction time real site weights. Also, the query time here is $O(b \log b)$ expected, while that of Cabello had multiple $\log n$ factors.

Theorem 5.5. (Voronoi diagram data structure) *Let H be a triangulated planar graph with a fixed embedding, and let S be a set of b sites placed on vertices on the boundary of its outer face with unspecified weights. We can construct in $O(nb^3W^2)$ time a data structure that supports the following kind of queries: given a generic and non-redundant set $S^1 \subset S$, whose weights are drawn from $[W]$, we can compute the Voronoi diagram of S^1 in $O(b \log b)$ expected or $O(b^2)$ worst-case time.*

5.1.3 Constructing the farthest-neighbor data structure

Before describing our farthest-neighbor data structure for Problem 5.1, we state the following lemma, taken almost verbatim from [Cab17b, Corollary 6]. Given a cycle γ in H, let

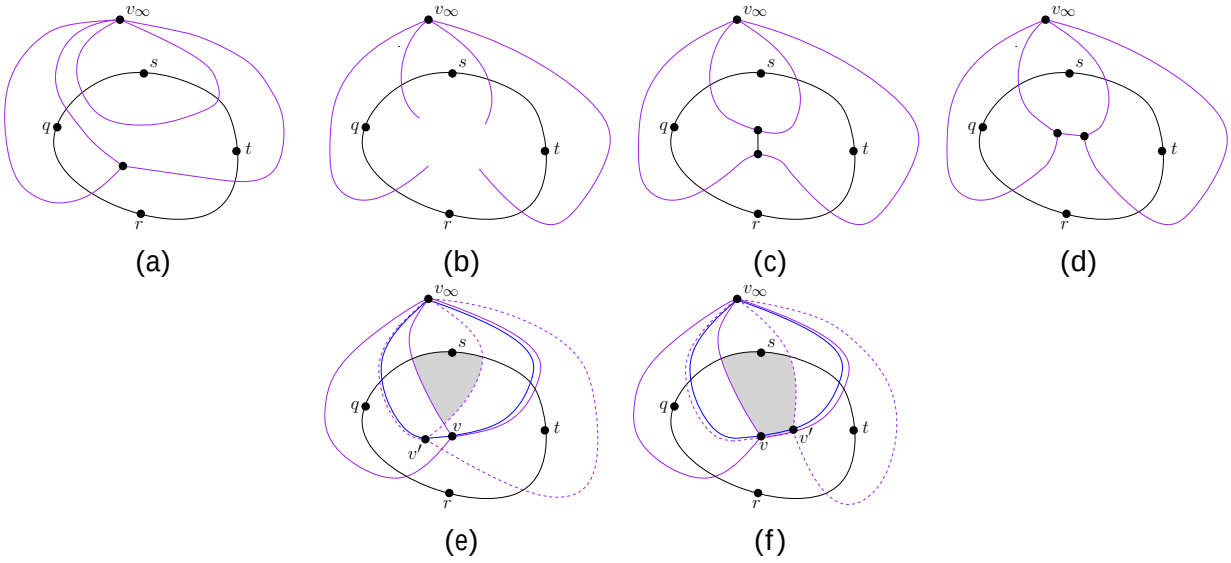


Figure 5.2: (a) A Voronoi diagram where a bisector bounds a Voronoi region. (b) Parts of the bisectors of consecutive sites when no bisector bounds a Voronoi region. (c)-(d) The Voronoi diagram for the two cases of (b). (e)-(f) The dual faces of the vertices in $\mathrm{VR}(s; t; q; r)$ (gray), $\mathrm{VD}(t, s; r; q)$ (bold), $\mathrm{VD}(t, s; r; t)$ (dotted), and $\mathrm{bis}(s; r)$ (blue), for the two cases of (b).

$V_{\text{int}}(\Pi; H)$ be the set of vertices of H enclosed by it. For a vertex $v_0 \in V$, a v_0-star-shaped cycle Π in H is a cycle that encloses the shortest path from v_0 to every $v \in V_{\text{int}}(\Pi; H)$.

Lemma 5.6. (Preprocessing for farthest-neighbor queries) *Let v_0 be a vertex of H. Assume that each vertex u of H has a cost $c(u) \geq 0$, and let $\Pi = \langle \pi_1, \ldots, \pi_k \rangle$ be a family of simple paths in H with a total of h edges, counted with multiplicity. After $O(n \log n)$ preprocessing time, we can answer the following kind of queries in $O(k \log h)$ time: given a v_0-star-shaped cycle Π in H, described as a concatenation of k subpaths from Π, return $\max_{u \in V_{\text{int}}(\Pi; H)} c(u)$.*

Proof Sketch. For each pair e_i and e_j of consecutive edges of a path in Π, we define $e_i; e_j$ to be the maximum cost of the vertices in the region "sandwiched" between the shortest paths from v_0 to an endpoint of the dual edges of e_i and e_j. Then, a query can be reduced to $O(k \log h)$ *range maximum queries* in the sequence of these $(i; j)$ numbers of each such path. See [Cab17b] for more details. □

Theorem 5.7. (Farthest neighbor) *We can construct a farthest-neighbor data structure for Problem 5.1 with $O(nb^3W^2 \log n)$ preprocessing time and $O(b^2 \log n)$ query time.*

Proof. Let S be a set of b sites as in Section 5.1.1, i.e., each $s \in S$ is placed at a vertex of X. The weights of the sites of S are unspecified. We construct the Voronoi diagram data structure of Theorem 5.5 and then apply Lemma 5.2 to compute all bisectors of each pair of sites. For each $s \in S$ we assign a *cost* $c_s(u) = \mathrm{dist}(v_s, u)$ to each $u \in U$ and $c_s(u) = 0$

to each $u \in V \setminus U$. Then, we construct the data structure of Lemma 5.6, where Π is the set of all bisectors $\mathrm{bis}(s, q)$, $\ell \leftarrow 6W$, $h \leftarrow 8nbW$, and $k \leftarrow b$. Note that for any $s, t \in S$, $\mathrm{bis}(s, t)$ is an s-star-shaped cycle in H because $\mathrm{VR}(s, \{s, t\}) \cup \mathrm{VR}(t, \{s, t\})$ is a connected subtree of the shortest-path tree of s with the same root. To avoid degeneracies, we arbitrarily order the edges of H that are incident to vertices of X and *perturb* the weight of the i-th such edge by adding to it a number ε^i, where $\varepsilon > 0$ is infinitesimal. We also compute the shortest path tree from every $x \in X$. The preprocessing time is $O(nb^3W^2)$.

In a query, we create naively in $O(b^2)$ time a set $S^1 \subseteq S$ by deleting from S each site s with $\ell(z_0, v_s) > \ell(z_0, v_t) + \mathrm{dist}(v_s, v_t)$ for some $t \in S$, ensuring that S^1 is non-redundant. Also the perturbation in the preprocessing guarantees that S^1 is generic. For each $s^1 \in S^1$ we set w_{s^1} equal to $\ell(z_0, v_{s^1})$ and then query the data structure of Theorem 5.5 to compute the Voronoi diagram of S^1 in $O(b^2)$ worst-case time (we do not need the faster $O(b \log b)$ randomized bound here). For each $s^1 \in S^1$ the boundary of $\mathrm{VR}(s^1, S^1)$ is the concatenation of at most b subpaths of the bisectors $\mathrm{bis}(s^1, q)$ because each Voronoi arc is a contiguous part of a bisector, as mentioned earlier. Thus, we can find $\max_{u \in \mathrm{VR}(s^1, S^1)} \mathrm{dist}(s^1, u)$ by employing Lemma 5.6. Finally, we return the maximum of these distances, for a total of $O(b^2)$ query time. □

5.2 Approximate diameter

Given a non-negatively-weighted, undirected planar graph $\mathcal{G}$ of N vertices and of diameter Δ, we show how to compute a $(1 - O(\varepsilon))$-approximation of Δ, which is equivalent to computing an $O(\varepsilon\Delta)$-additive approximation. Let $\tilde{\Delta}$ be a 2-approximation of Δ, which we can compute in linear time [HKRS97].

We adapt the recursive scheme of Weimann and Yuster [WY16]. The input to our algorithm is a non-negatively-weighted, undirected planar graph G whose vertices are either marked or unmarked, and the output is an $O(\varepsilon\Delta)$-additive approximation of the longest shortest-path distance of any two marked vertices of G. In the first recursive call, we have $G = \mathcal{G}$, all vertices of G are marked, and $n = N$. Let G_1 and G_2 be two of G's subgraphs such that each marked vertex of G lies in at least one of them. We denote by $d(G_1, G_2, G)$ the longest shortest-path distance in G between a marked vertex in G_1 and another in G_2. Notice that $d(G, G, G) = \max\{d(G_1, G_2, G), d(G_1, G_1, G), d(G_2, G_2, G)\}$. We make the following assumption for the distances between marked vertices of G, which states the distance in G between any marked vertices is an $O(\varepsilon\Delta)$-additive approximation of their distance in $\mathcal{G}$.

Assumption 5.1. (Distances between marked vertices) *For every two marked vertices s, t of G, we have $\mathrm{dist}_G(s, t) \le \mathrm{dist}_G(s, t) \le \mathrm{dist}_G(s, t) + \varepsilon\Delta$.*

Recall that a *separator* of a planar graph is a subset of its vertices whose removal decomposes it into at least two disjoint, induced subgraphs. If the size of each is at most

a constant fraction of that of the original graph, the separator is said to be *-balanced*. Moreover, if the vertices of the separator are the vertices of shortest paths with common root, it is called *shortest-path* separator.

Our algorithm performs the following steps.

1. Find an -balanced shortest-path separator C of G, for some constant 1, such that the removal of its vertices decomposes G into two disjoint subgraphs A and B. Let G_{in} $A \cup C$ and G_{out} $B \cup C$. See Section 5.2.1.
2. Compute an $O(\varepsilon'' \Delta)$-additive approximation of $d(G_{\text{in}}, G_{\text{out}}, G)$. See Section 5.2.2.
3. Unmark each vertex of C in both G_{in} and G_{out} (these vertices appear in both graphs, so we have already considered all such pairs of marked vertices). *Augment* G_{in} with extra (unmarked) vertices and edges into a non-negatively-weighted, undirected planar graph G_{in} which satisfies Assumption 5.1 and its size is roughly the same as the number of marked vertices of G_{in}. Construct a similarly defined graph G_{out} for G_{out} and recursively solve the problem in G_{in} and G_{out}. See Section 5.2.3.
4. Return max $d(G_{\text{in}}, G_{\text{out}}, G)$; $d(G_{\text{in}}, G_{\text{in}}, G_{\text{in}})$; $d(G_{\text{out}}, G_{\text{out}}, G_{\text{out}})$ (.

5.2.1 Decomposing G

As explained in Section 2.5, to compute the shortest-path separator C in Step 1 we first find the shortest-path tree T from any marked vertex w_0 of G in linear time [HKRS97]. There are two root paths R and Q in T, which can be computed in linear time, such that the removal of their vertices decomposes G into two disjoint planar subgraphs A and B of at most $2n/3$ vertices each. However, the size of C $R \cup Q$ can be as big as n. See Figure 5.3(a)-(b). Let G_{in} $A \cup C$ and G_{out} $B \cup C$.

5.2.2 Approximating $d(G_{\text{in}}, G_{\text{out}}, G)$

Let M_{in} (respectively M_{out}) be the set of marked vertices of G_{in} (respectively G_{out}). We want to $O(\varepsilon'' \Delta)$-additively approximate the distance from each $v \in M_{\text{out}}$ to its farthest neighbor in M_{in} (i.e., $\max_{u \in M_{\text{in}}} \min_{c \in C} \{dist_G(v,c) + dist_G(c,u)\}$), and return the maximum. To do that, we would like to construct the farthest-neighbor data structure of Theorem 5.7 with H G_{in}, X C, and U M_{in}. Then, for each $v \in M_{\text{out}}$ we want to query that data structure with z_0 v, X^1 X, and (z_0, x^1) $dist_G(z_0, x^1)$ for each $x^1 \in X^1$. However, there are two issues with that approach. First, C could have $O(n)$ vertices, thus leading to superlinear total time. Second, the edge weights (z_0, x^1), where $x^1 \in X^1$, for each query are not necessarily small integers (because the distances $dist_G(z_0, x^1)$ are in general non-negative numbers).

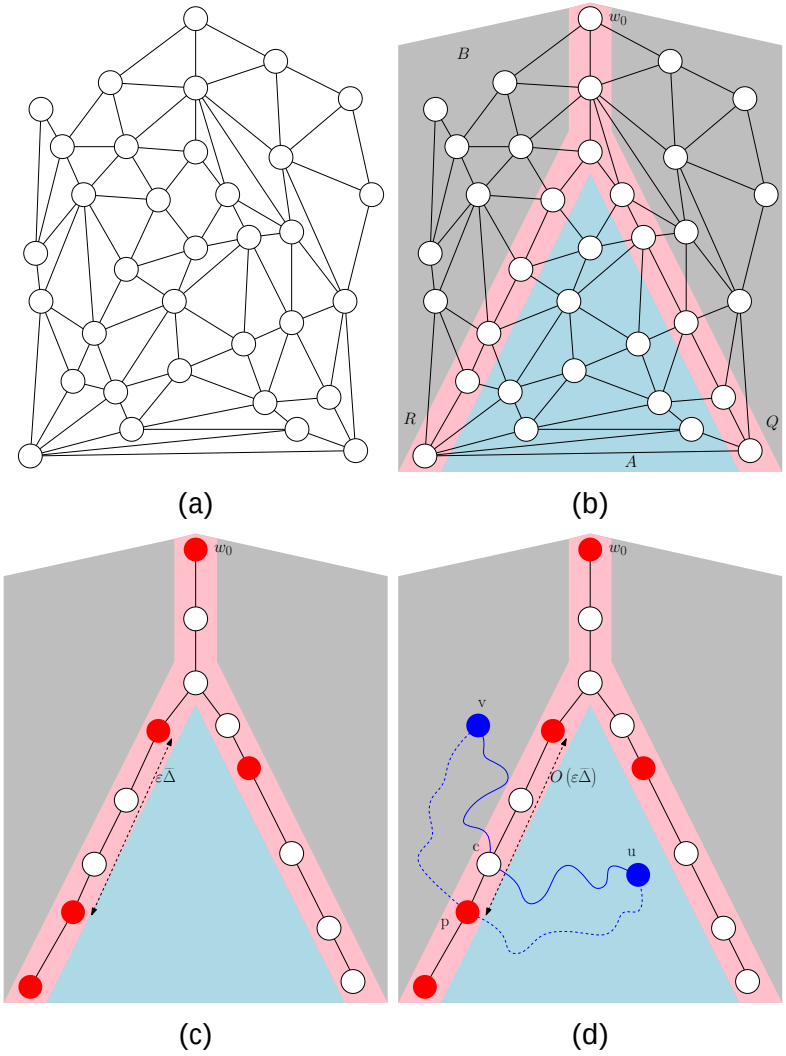


Figure 5.3: (a) A planar graph. (b) A shortest-path separator (its vertices on the pink background) and the resulting decomposition. The vertices of A are on the blue background (similarly for B). The vertices of G_{in} are on the blue and pink background (similarly for G_{out}). (c) The portals (red vertices). (d) Detour through portals.

For the first issue, we show that by increasing the error by $O(\varepsilon\Delta)$, it suffices to choose only a small subset of the vertices of C. Specifically, we build a set P of $O(1/\varepsilon)$ vertices, called *portals*, and route any v-to-u shortest path through them, where $v \in M_{\text{out}}$ and $u \in M_{\text{in}}$. To construct P, we start from the root of C and perform a linear walk in the $8\tilde{\Delta}$-prefix of both R and Q (as mentioned above, $\tilde{\Delta}$ is a 2-approximation of Δ). Notice that the diameter of G can be much bigger than Δ because when we recursively solve the problem in G_{in} and G_{out} only the distances between marked vertices are approximately preserved. With these walks, we select $O(1/\varepsilon)$ portals on each path, such that any consecutive pair of them is at distance $\varepsilon\Delta$ thereon and discard any duplicates. See Figure 5.3(c). Let

$d^{\perp}(G_{\text{in}}, G_{\text{out}}, G) \quad \max_{v \in M_{\text{out}}, u \in M_{\text{in}}} \min_{p \in P} \{dist_G(v,p) + dist_G(p,u)\}$.

Lemma 5.8. (Approximation with portals) *We have* $d(G_{\text{in}}, G_{\text{out}}, G) \le d^{\perp}(G_{\text{in}}, G_{\text{out}}, G) \le d(G_{\text{in}}, G_{\text{out}}, G) + \varepsilon\Delta$.

Proof. Since $d^{\perp}(G_{\text{in}}, G_{\text{out}}, G)$ corresponds to a path in G, we have that $d(G_{\text{in}}, G_{\text{out}}, G) \le d^{\perp}(G_{\text{in}}, G_{\text{out}}, G)$. Let $v \in M_{\text{out}}$ and $u \in M_{\text{in}}$ be such that $dist_G(u,v) = d(G_{\text{in}}, G_{\text{out}}, G)$. Recall that from Assumption 5.1 we have $dist_G(s,t) \le dist_G(s,t) \le dist_G(s,t) + \varepsilon\Delta$ for any two marked vertices s, t of G. Since $dist_G(s,t) \le \Delta$, we have $dist_G(s,t) \le (1+\varepsilon)\Delta$.

Let $c \in C$ be a vertex that the v-to-u shortest path in G crosses and assume without loss of generality that $c \in R$. We now show that c lies in 8Δ-prefix of R. All of v, u, and w_0 are marked, so the w_0-to-v and the v-to-u shortest paths are of length at most $(1+\varepsilon)\Delta$. Since c lies on the v-to-u shortest path, the length of the v-to-c shortest path is also upper-bounded by $(1+\varepsilon)\Delta$. Hence, the concatenation of the w_0-to-v and the v-to-c shortest paths has length $2(1+\varepsilon)\Delta$, which is indeed smaller than 8Δ for $\varepsilon \le 1$ (since $\Delta \le 2\Delta$).

Let $p \in P$ be the portal on R that is closest to c. From the way we built P, we have $dist_G(p,c) \le \varepsilon\Delta$. The triangle inequality implies that $dist_G(v,p) \le dist_G(u,c) + dist_G(c,p) \le dist_G(u,c) + \varepsilon\Delta$ and $dist_G(p,u) \le dist_G(c,u) + \varepsilon\Delta$. Using these inequalities, we have $d^{\perp}(G_{\text{in}}, G_{\text{out}}, G) \ge dist_G(v,p) - dist_G(p,u) \ge dist_G(v,c) - dist_G(c,u) - 2\varepsilon\Delta \ge dist_G(v,u) - 2\varepsilon\Delta \ge d(G_{\text{in}}, G_{\text{out}}, G) - 2\varepsilon\Delta$. See Figure 5.3(d). □

For the second issue, we show how to ensure that whenever we query the data structure of Theorem 5.7 each $\ell(z_0, x)$ ($x \in X$) is a small integer. We assume that without loss of generality $1/\varepsilon$ is an integer. First, we compute the shortest-path tree in G from every $p \in P$ in $O((1/\varepsilon)n)$ time [HKRS97] and create a value $\hat{d}(v,p)$ for each $v \in M_{\text{out}}$ and $p \in P$ by first rounding $dist_G(v,p)$ to the closest multiple of $\varepsilon\Delta$ and then dividing it with that number. If $dist_G(v,p) \ge 4\Delta$, then $dist_G(v,p)$ will be irrelevant in approximating $d^{\perp}(G_{\text{in}}, G_{\text{out}}, G)$. Thus, in this case we set $\hat{d}(v,p) = 4/\varepsilon$. Notice that now $\hat{d}(v,p) \in \{0, \ldots, 4/\varepsilon\}$. We divide (but not round) every edge weight of G_{in} by $\varepsilon\Delta$ and denote the resulting graph by $\hat{G}_{\text{in}}$. Let $\hat{d} = \varepsilon\Delta \cdot \max_{v \in M_{\text{out}}, u \in M_{\text{in}}} \min_{p \in P} \{\hat{d}(v,p) + dist_{\hat{G}_{\text{in}}}(p,u)\}$. Thus, we have $d(G_{\text{in}}, G_{\text{out}}, G) \le \hat{d} \le d(G_{\text{in}}, G_{\text{out}}, G) + \varepsilon\Delta$.

We can finally construct the farthest-neighbor data structure of Theorem 5.7 with $H = \hat{G}_{\text{in}}$, $X = P$, $U = M_{\text{in}}$, $b = O(1/\varepsilon)$, and $W = 4/\varepsilon$. For each $v \in M_{\text{out}}$, we query the data structure with $z_0 = v$, $X = P$, and $\ell(z_0, x) = \hat{d}_G(v,x)$, where $x \in X$, and multiply the answer by $\varepsilon\Delta$. The total time to $O(\varepsilon)$-additively approximate $d(G_{\text{in}}, G_{\text{out}}, G)$ is $O(nb^3W^2 + nb^2) = O((1/\varepsilon)^5 n)$.

Contrary to our approach, Weimann and Yuster [WY16] employed a brute-force search, after observing that there are only $2^{O(1/\varepsilon)}$ combinatorially different vertices of M_{in} and M_{out} (in terms of their vectors of distances to the portals).

5.2.3 Recursively solving the problem in G_{in} and G_{out}

The vertices on C appear in both G_{in} and G_{out}, so we have already considered all such pairs of marked vertices and can unmark them in both graphs. Then, we need to augment G_{in} into a graph G_{in} to ensure that Assumption 5.1 is satisfied, i.e., $dist_G(s,t) = dist_{G_{\text{in}}}(s,t) = dist_G(s,t)$ for any two marked vertices s,t of G_{in}. Also, we need to ensure that the size of G_{in} is roughly the same as the number of marked vertices of G_{in}. We want to augment G_{out} to a similarly defined graph G_{out} and use recursion in G_{in} and G_{out}.

We start at the common root of R and Q and select with a linear walk on their 8ε-prefices $1\{\varepsilon'$ of their vertices, called *dense portals*, where $\varepsilon' \in \varepsilon$ is a parameter to be set later, such that any consecutive pair of them is at distance $\varepsilon'\Delta$ thereon. The union B_{in} of the $p1\{\varepsilon'q^2$ shortest paths in G_{out} of every pair of dense portals has at most $p1\{\varepsilon'q^4$ vertices of degree more than two. That is guaranteed by generating these shortest paths with the data structure of Klein [Kle05], which ensures that any two of them share at most a common subpath (see Theorem 5.10). Thus, we can shrink the rest to obtain a planar graph B^1_{in}. Then, we unmark the vertices of B^1_{in} and create a new graph $G^1_{\text{in}} = G_{\text{in}} \cup B^1_{\text{in}}$ of $|V(G)|$ $p1\{\varepsilon'q^4$ vertices. Its size can be reduced to $p1\{\varepsilon'q^4$, where $\varepsilon \geq 2|V(G)|\{3$ is the set of vertices of $V(G_{\text{in}}) \cup V(C)$. To do so, we delete each non-dense-portal c and redirect each edge cu, where $u \in V(G_{\text{in}}) \cup V(C)$, from c to its closest dense portal p and change its weight to $dist_G(p,u)$. We also create an edge between every consecutive pair s,t of dense portals on R and Q with weight $dist_G(s,t)$. Notice that all these edges can be created without affecting the planarity. Let G_{in} be the ensuing graph. See Figure 5.4.

Lemma 5.9. (Recursion with dense portals) *For any two marked vertices u,v of G_{in}, we have $dist_G(s,t) \leq dist_{G_{\text{in}}}(s,t) \leq dist_G(s,t) + O(\varepsilon'\Delta)$.*

Proof. A path in G_{in} corresponds to a path in G, and u and v are marked in both G and G_{in}. Thus, $dist_G(u,v) \leq dist_{G_{\text{in}}}(u,v)$. If (u,v) uses only edges incident to vertices in $V(G_{\text{in}}) \cup V(C)$, then it also exists in G_{in}, and we trivially have $dist_G(u,v) = dist_{G_{\text{in}}}(u,v)$.

Otherwise, we assume without loss of generality that (u,v) is composed of the u-to-c shortest path in G_{in}, the c-to-c^1 shortest path in G (which either lies entirely in G_{in} or not), and the c^1-to-v shortest path in G_{in}, where $c \in R$ and $p \in Q$. As in the proof of Lemma 5.8, we can show that c and c^1 lie in the 8ε-prefix of R and Q respectively, i.e., $dist_G(w_0,c) + dist_G(w_0,c^1) \leq 8\varepsilon$, where w_0 is the common root of P and Q. Let p and p^1 be the closest dense portals to c and c^1 respectively, so $dist_G(c,p) + dist_G(c^1,p^1) \leq \varepsilon'\Delta$. From the way we deleted the non-dense-portals and redirected their incident edges, there is a u-to-p path in G_{in} of length at most $dist_G(u,c) + dist_G(c,p) \leq dist_G(u,c) + \varepsilon'\Delta$ and a p^1-to-v path of length at most $dist_G(p^1,c^1) + dist_G(c^1,v) \leq dist_G(c^1,v) + \varepsilon'\Delta$.

If the c-to-c^1 shortest path in G does not lie entirely in G_{in}, then from the way we constructed B_{in}, we know $dist_{G_{\text{in}}}(p,p^1) \leq dist_{G_{\text{out}}}(c,c^1) \leq dist_{G_{\text{out}}}(p,c) + dist_{G_{\text{out}}}(p^1,c^1) \leq dist_G(c,c^1) + 2\varepsilon'\Delta$. Otherwise, let the c-to-c^1 shortest path in G be composed of the c-to-x

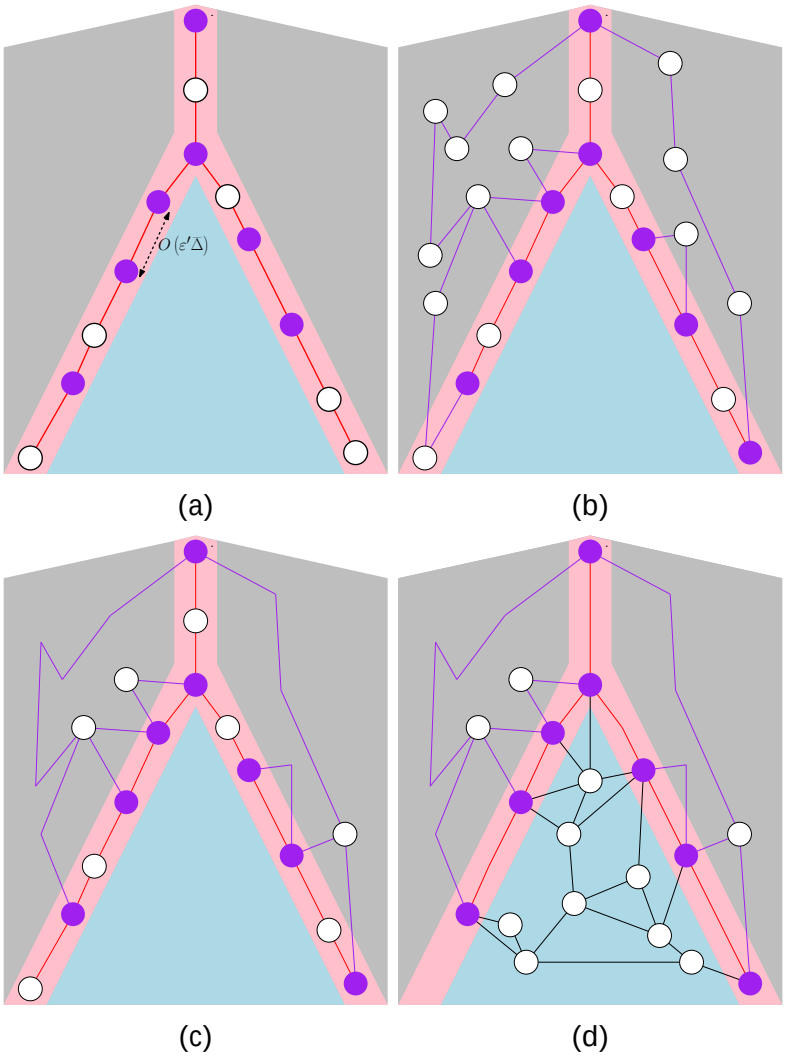


Figure 5.4: (a) The dense portals (purple vertices) for the planar graph of Figure 5.3(a). (b)-(c) The graphs B_{in} and B^1_{in}, respectively (on the grey background). (d) The graph G_{in}.

shortest path in R, the x-to-x^1 shortest path in $V(G_{\text{in}}) \setminus V(C)$, and the x^1-to-c^1 shortest path in Q, where $x \in R$ and $x^1 \in Q$. We can similarly show that in this case we also have $dist_{G_{\text{in}}}(p, p^1) \le dist_G(c, c^1) + 2\varepsilon''^1\Delta$.

Therefore, the concatenation of the u-to-p, p-to-p^1, and p^1-to-v shortest paths in G_{in} yields a path of length $dist_{G_{\text{in}}}(u, p) + dist_{G_{\text{in}}}(p, p^1) + dist_{G_{\text{in}}}(p^1, v) \le dist_{G_{\text{in}}}(u, c) + dist_G(c, c^1) + dist_{G_{\text{in}}}(c^1, v) + O(\varepsilon''^1\Delta) \le dist_G(u, v) + O(\varepsilon''^1\Delta)$. □

It remains to show how to construct B^1_{in} and how to set ε''^1. For the first, contrary to Weimann and Yuster, who constructed B^1_{in} explicitly in $O(\varepsilon^{-1}n)$ time, we employ a method based on the multiple shortest paths data structure of Klein [Kle05].

Theorem 5.10. (Augmented graph) *We can build B^1_{in} in $O(n \log n)$ $(1/\varepsilon)^4 \log n)$ time.*

Proof. As Klein showed, we can construct an implicit representation of the shortest-path tree $T(u)$ of each vertex u on the boundary of the outer face of G_{in} in $O(n \log n)$ total time. The order of the children $w_1, w_2, \ldots, w_k$ for some $k \geq 0$, of a vertex v in $T(u)$ is specified as follows: w_i is left of w_j if and only if vw_i occurs in G_{in} strictly between vw_i and vp in a counterclockwise traversal that starts at vp, where p is v's parent in $T(u)$. Klein used a *persistent* [DSST89] version of *dynamic trees* [ST83] to represent the $T(u)$'s, so we can find the u-to-v shortest-path distance, for any $v \in V(G_{\text{in}})$, in $O(\log n)$ time. We want to augment Klein's data structure to also support the following two queries on each $T(u)$: (i) find the lowest common ancestor of any two vertices; and (ii) find the level ancestor of any vertex and any level. We can accommodate both queries in $O(\log n)$ time by merely replacing the dynamic trees with the (persistent) *top-tree* structures of Alstrup et al. [AHLT05].

To build B^1_{in}, we construct the modified version of Klein's data structure for G_{out}, after redrawing it in linear time (if needed), such that the vertices of C lie on the outer face, and properly query it to find all vertices of G_{out} of degree more than two in B_{in}, which, as argued before, are $O((1/\varepsilon)(1/\varepsilon)^4)$. There are three cases for each pair of shortest paths between dense portals: they (i) do not intersect, (ii) intersect only at one vertex, or (iii) share a common subpath. For any four dense portals a, b, c, and d, assuming without loss of generality that the latter case holds for $\gamma_G(a, b)$ and $\gamma_G(c, d)$ and that c is between a and b on the boundary of the outer face of G_{in}, we want to find the first and the last vertices, p_1 and p_2, on that subpath.

We find p_1 with a binary search on $\gamma_G(a, b)$ and T_c (finding p_2 is similar). Let p^1 and p^2 be initially set to a and b respectively, and let p be the vertex midway between p^1 and p^2 on $\gamma_G(a, b)$, which can be found by a level ancestor query on T_a. We want to determine whether p is (i) between p_1 and p_2 (i.e., on $\gamma_G(c, d)$), (ii) between a and p_1, or (iii) between p_2 and b. To do so, we find the lowest common ancestor lca of p and d on T_c. If $lca = p$, we are in case (i) because p is on $\gamma_G(c, d)$. See Figure 5.5(a) and (c). Else, we perform a level ancestor query for p and d to find the children $\hat{p}$ and $\hat{d}$ of lca that lie on $\gamma(p, lca)$ and $\gamma(d, lca)$, respectively, and compare their order around lca. If $\hat{p}$ is to the left of $\hat{d}$, we are in case (ii), else we are in case (iii). See Figure 5.5(b) and (d). For case (i) or (iii) we recurse with $p^1 = p$, for case (ii) with $p^2 = p$, and we stop when $p^1 = p^2$. Once we have processed every pair of dense portals, we shrink the vertices of $V(G_{\text{out}})$ of degree at most two with a linear scan, thus obtaining B^1_{in}. □

Finally, Weimann and Yuster used a fixed constant for ε_1 throughout their algorithm. There are $O(\log N)$ recursion levels, and the error accumulates, hence they set $\varepsilon_1 = \varepsilon / \log N$. However, with that choice of ε_1, the four $O((1/\varepsilon_1))$ factors of the size of B^1_{in} lead to four $O(\log N)$ factors in the running time of their solution. Instead, we make ε_1 adaptive (i.e., dependent on the current input size n), namely equal to $\varepsilon / n^{1/8}$, thus shaving off two $\log N$ factors, while retaining the $1 + O(\varepsilon)$ approximation factor, as we show next.

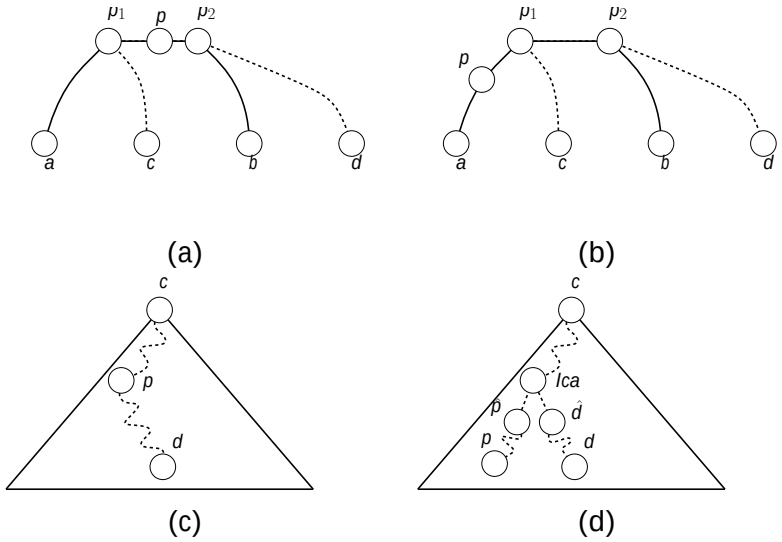


Figure 5.5: (a), (c) p lies between p_1 and p_2. (b), (d) p lies between a and p_1.

5.2.4 Analyzing our algorithm

Approximation factor. As described in Section 5.2.2, for each node of the recursion tree we compute an $O(\varepsilon'\Delta)$-additive approximation of $d(G^{p,q}_{in}; G^{p,q}_{out}; G^{p,q})$, where $G^{p,q}$ is the graph associated with , and $G^{p,q}_{in}$ and $G^{p,q}_{out}$ are the two graphs created by decomposing $G^{p,q}$, as in Section 5.2.1. As explained in Section 5.2.3, from the way we apply recursion, in each child of we have $d(G^{p,q}_{in}; G^{p,q}_{out}; G^{p,q}) \le d(G^{p,q}_{in}; G^{p,q}_{out}; G^{p,q}) + O(\varepsilon'^{1}\Delta)$ and $|V(G^{p,q}_{in})| \le (2/3)|V(G^{p,q}_{in})| + O(\log(1/\varepsilon)\varepsilon^{-4})$, where $\varepsilon'^{1} = \varepsilon'/(M|V(G^{p,q}_{in})|^{1/8})$. At the root of the recursion tree, we have $G^{p,q} = \mathcal{G}$, while in each leaf , $|V(G^{p,q})| = \Theta(\log(1/\varepsilon)\varepsilon^{-4})$.

Thus, the additive error of our algorithm is $O(\varepsilon'\Delta) + O(\sum_i \varepsilon'_i\Delta)$, where $\varepsilon'_i = \varepsilon'(n_i)^{1/8}$ for some sequence $n_1, n_2, \ldots, n_k$ that satisfies $n_{i+1} \le (3/4)n_i \le 2n_{i+1}$ and $n_k = \Theta(\log(1/\varepsilon)\varepsilon^{-4})$ with $n_1 = N$. Since n_i decreases at least exponentially, ε'_i grows likewise. Hence, $\sum_i \varepsilon'_i$ is similar to a geometric series and can be upper-bounded by the last term, which is $O(\varepsilon'^{3/2})$. We conclude that the additive error is $O(\varepsilon'\Delta)$, implying that the approximation factor of our algorithm is $1 + O(\varepsilon')$. That can be refined to $1 + \varepsilon'$ after adjusting ε' by a constant factor.

Assumption 5.1 is true. We now show that for any two marked vertices u and v of a graph G encountered during the recursion, we have $dist_G(u,v) \le dist_G(u,v) \le dist_G(u,v) + \varepsilon'\Delta$.

Fix a non-leaf node of $\mathcal{T}$ and its parent , such that u and v are marked in both and $G^{p,q}$ G. As explained in Section 5.2.3, from the way we use recursion, we have

$dist_{G'}(u,v) \le dist_{G'}(u,v) + dist_{G'}(u,v) + dist_{G'}(u,v) \le O(\varepsilon^{-1}\Delta)$, where $\varepsilon_1 = \varepsilon/8$. At the root of the recursion tree, $G' = G$, while in each leaf, $|V(G')| = O(1/\varepsilon^4)$.

Thus, we have that $dist_G(u,v) \le dist_G(u,v) + dist_G(u,v) + dist_G(u,v) \le (1+\varepsilon_i)\Delta$, where $\varepsilon_i = \varepsilon/8$, for some sequence $n_1, n_2, \ldots, n_k$ that satisfies $n_{i+1} \le n_i \le 2n_{i+1}$ and $O(1/\varepsilon^4) \le n_i \le n$ with $n_1 = n$ and $n_k = \Theta(1/\varepsilon^4)$. As above, $\sum_i \varepsilon_i$ can be upper-bounded by $O(\varepsilon)$. Hence, for any two marked vertices u and v of G', we have $dist_G(u,v) \le dist_G(u,v) + dist_G(u,v) \le (1+O(\varepsilon))\Delta$, thus proving (a stronger version of) the assumption.

Running time. In a graph of size n, the running time $T(n)$ of our algorithm satisfies the following inequality:

$$T(n) \le \max_{1/3 \le \alpha \le 2/3} T(\alpha n + O(1/\varepsilon^4)) + T((1-\alpha)n + O(1/\varepsilon^4)) + O(n \log n \log(1/\varepsilon^5)).$$

In the base case, there are $O(1/\varepsilon^4 N)$ graphs of $O(1/\varepsilon^4)$ vertices each, so we need $O(1/\varepsilon^4 N \log(1/\varepsilon^4 N))$ time [Fre87]. For $n \ge N$, we have $T(N) \le O(N \log N \log N \log(1/\varepsilon^5)) \le (1/\varepsilon^5)$, which is the total time of our algorithm.

New Result 5. (Approximate diameter in weighted, undirected planar graphs) *We can compute a $(1+\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph of n vertices in $O(n \log n \log(1/\varepsilon))$ time.*

Remarks:

- Gawrychowski et al. [GKM+18] recently improved Cabello's algorithm [Cab17b] for computing the diameter in planar graphs exactly. Their algorithm is deterministic instead of randomized and requires $\tilde{O}(n^{5/3})$ time instead of $\tilde{O}(n^{11/6})$. It is likely that their techniques can be used to shave off some $1/\varepsilon$ factors.
- An interesting consequence of our result is that we can compute the *exact* diameter of an *unweighted* planar graph in $O(n \log n \log n)$ $= \Delta^{O(1)}$ time, where Δ is the diameter, simply by setting ε near $1/\Delta$. If one wants running time near linear in n, the best previous result we are aware of was by Eppstein [Epp99] and had exponential dependence in Δ (namely, the time bound is $O(n2^{O(\log)})$). Note that our result beats Cabello's or Gawrychowski et al.'s algorithm when the diameter is smaller than n for some constant .

By keeping track throughout the algorithm of the distance of each vertex to its farthest neighbor, we can compute a $(1+\varepsilon)$-approximation of its eccentricity. Hence, we can also compute a $(1+\varepsilon)$-approximation of the radius (i.e., the minimum eccentricity) of the graph.

Corollary 5.11. (Approximate eccentricities, farthest neighbors, and radius in weighted, undirected planar graphs) *Given a non-negatively weighted, undirected planar graph of n vertices, we can compute a $(1+\varepsilon)$-approximation of the radius of the graph and of the eccentricity of each node (and an approximate farthest neighbor) in $O(n \log n (\log n + 1/\varepsilon^5))$ time.*

5.3 Approximate distance oracles

To construct an approximate distance oracle in the Word RAM, we build upon the general framework of the oracles of Kawarabayashi et al. [KST13] and of Gu and Xu [GX15]. Specifically, given a non-negatively-weighted, undirected planar graph $\mathcal{G}$ of N vertices and of diameter Δ, we focus on constructing a distance oracle with additive stretch $O(\varepsilon\Delta)$, i.e., a data structure that supports the following kind of queries: given for any two vertices u and v of $\mathcal{G}$, return a value $\tilde{d}$ with $dist_{\mathcal{G}}(u,v) \le \tilde{d} \le dist_{\mathcal{G}}(u,v) + O(\varepsilon\Delta)$. Then, we can obtain a $(1+\varepsilon)$-approximate distance oracle with an approach based on *sparse neighborhood covers*, defined in Section 2.7. Let $\tilde{\Delta}$ be a 2-approximation of Δ.

5.3.1 Distance oracles with additive stretch

The decomposition tree. We recursively decompose $\mathcal{G}$, as in Sections 5.2.1 and 5.2.3, but now we also store the ensuing graphs in a *recursive decomposition tree* $\mathcal{T}$ with the following properties.

- $\mathcal{T}$ has degree two and height $O(\log N)$.
- Each non-leaf node $\quad$ of $\mathcal{T}$ is associated with a graph $G^{(\quad)}$. The graph of each child of $\quad$ has at most $2|V(G^{(\quad)})|/3 + O((1/\varepsilon')^4)$ vertices, where $\varepsilon' = \varepsilon/|V(G^{(\quad)})|/8$. At the root $\quad$ of $\mathcal{T}$, $G^{(\quad)} = \mathcal{G}$, and at each leaf, $|V(G^{(\quad)})| = O((1/\varepsilon')^4)$.
- Each non-leaf node $\quad$ of $\mathcal{T}$ is also associated with a shortest-path separator $C^{(\quad)}$. We call a vertex of $G^{(\quad)}$ *marked* if and only if it does not lie in $C^{(\quad)}$ of any ancestor $\quad$ of $\quad$. Each marked vertex of $G^{(\quad)}$ that is not on $C^{(\quad)}$ is contained in the graph of one child of $\quad$.
- Fix a non-leaf node $\quad$ of $\mathcal{T}$ and let $\quad$ be its parent. For each pair of marked vertices of $G^{(\quad)}$, we have $dist_{G^{(\quad)}}(u,v) \le dist_{G^{(\quad)}}(u,v) \le dist_{G^{(\quad)}}(u,v) + O(\varepsilon'\Delta)$.

Our data structure. In each non-leaf node $\quad$ of $\mathcal{T}$, we find in linear time, as in Section 5.2.2, a set $P^{(\quad)}$ of $1/\varepsilon'$ portals on $C^{(\quad)}$, such that for any two marked vertices $u, v \in V(G^{(\quad)})$ on different sides of $C^{(\quad)}$,

$dist_{G^{p\ q}}(u,v) \leq \min_{p \in P^{p\ q}} \{dist_{G^{p\ q}}(u,p) + dist_{G^{p\ q}}(p,v)\} \leq dist_{G^{p\ q}}(u,v) + O(\varepsilon)\Delta$.

Then, we run an SSSP algorithm from each $p \in P^{p\ q}$. Let $_1$ and $_2$ be the children of . For every marked vertex v of $_1$, we create a value $\hat{d}_{G^{p\ q}}(v,p) \in (1+\varepsilon)$ as in Section 5.2. We also divide every edge weight of $G^{p\ q}$ by $\varepsilon\Delta$. Thus,

$dist_{G^{p\ q}}(u,v) \leq \varepsilon\Delta \cdot \min_{p \in P^{p\ q}} \{\hat{d}_{G^{p\ q}}(u,p) + dist_{G^{p\ q}}(p,v)\} \leq (1+\varepsilon) \cdot dist_{G^{p\ q}}(u,v) + O(\varepsilon)\Delta$.

We create a set S of $O(1/\varepsilon)$ sites with unspecified weights in $G^{p\ 2q}$, place each at a vertex of P. Also, we perturb the weights of the edges incident to the sites as in Section 5.2.2. Then, we construct for $G^{p\ 2q}$ and S a data structure for Voronoi diagrams, similarly to that of Theorem 5.5, in $O(n \log^5(1/\varepsilon))$ time. For each marked vertex v of $_1$, we construct an empty set P^1 and insert to it every $p \in P^{p\ q}$, such that the last edge on the v-to-p shortest path in $G^{p\ q}$ does not lie in $G^{p\ 2q}$, which we can determine by inspecting p's shortest-path tree. Next, we query the data structure of Theorem 5.5 to construct the Voronoi diagram in $G^{p\ 2q}$ of a set $S^1 \subseteq S$ of sites placed at the vertices of P^1, where the weight of each $s^1 \in S$ is equal to $\hat{d}_G(v, v_{s^1}s)$. Notice that from the perturbation and from the choice of P^1, we have that S^1 is generic and non-redundant. Finally, we build for that Voronoi diagram the *vertex location* data structure of [GMWWN18, Section 6]. That data structure preprocesses once in $O(b \log n)$ time and space a planar graph of n vertices and a set of b sites with unspecified weights that lie on the boundary of its outer face to support the following operation: preprocess any given Voronoi diagram of a subset of these sites in $O(b)$ time and space, such that the site that contains a given vertex can be found in $O(\log b)$ time. In our application, $b \leq 1/\varepsilon$, so we need $O(n/\varepsilon)$ total time and space for preprocessing all Voronoi diagrams related to the marked vertices of $G^{p\ 1q}$.

Given two vertices u and v, we find in $O(1)$ time the lowest node of $\mathcal{T}$ where both are marked (that can be done with linear preprocessing time). If none exists, u and v must reside in a leaf, and we return their shortest-path distance by looking up the distance matrix therein (which we can precompute). Else, we assume without loss of generality that $u \in V \ G^{p\ 2q}$ and $v \in V \ G^{p\ 1q}$, where $_1$ and $_2$ are the children of , and that we have computed the Voronoi diagram in $G^{p\ 2q}$ for sites and weights prescribed by v. Then, we query, in $O(\log(1/\varepsilon))$ time, the vertex location data structure for that Voronoi diagram to find the site p therein whose Voronoi region contains u. Finally, we return $d' \quad \varepsilon\Delta \cdot (dist_{G^{p\ q}}(v,p) \quad dist_{G^{p\ 2q}}(p,u))$.

Additive stretch. We now show that for any two vertices u, v of $\mathcal{G}$, our oracle returns a value d' with $dist_G(u,v) \leq d' \leq dist_G(u,v) \quad O(\varepsilon)\Delta$.

Fix a non-leaf node of $\mathcal{T}$ and its parent , such that u and v are marked in both. From the way we use recursion, we have $dist_{G^{p\ q}}(u,v) \leq dist_{G^{p\ q}}(u,v) \leq dist_{G^{p\ q}}(u,v)$ $O(\varepsilon'^{-1}\Delta)$, where $\varepsilon' \quad \varepsilon/(|V(G^{p\ q})|^{18})$. At the root of the recursion tree, $G^{p\ q} \quad \mathcal{G}$, while at each leaf , $\left|V \quad G^{p\ q}\right| \quad \Theta(1/\varepsilon^4)$. Also, from the way we approximate $dist_{G^{p\ q}}(u,v)$, we have $dist_{G^{p\ q}}(u,v) \leq d' \leq dist_{G^{p\ q}}(u,v) \quad O(\varepsilon)\Delta$.

Thus, the value d' that our oracle returns is such that $dist_G(u,v) \leq d' \leq dist_G(u,v) \cdot O(\varepsilon) + O(\varepsilon) \cdot \Delta$, where $\varepsilon_i = \varepsilon/n_i^{18}$, for some sequence $n_1, n_2, \ldots, n_k$ that satisfies $n_{i-1}^3 \leq n_i \leq 2n_{i-1}^3$ and $O(\log^* n) + \sum_i \varepsilon_i$ with $n_1 = N$ and $n_k = \Theta(\log^* n)$. As in Section 5.2.4, $\sum_i \varepsilon_i$ can be upper-bounded by $O(\varepsilon^{3/2})$, hence yielding the lemma.

Preprocessing time and space. The preprocessing time $T(n)$ and space $S(n)$ satisfy the following recurrence relations:

$$T(n) \leq \max_{1/3 \leq \alpha \leq 2/3} T(n^\alpha) \cdot O(\varepsilon^{-4} \cdot n^{1-\alpha}) + T(\alpha n) \cdot O(\varepsilon^{-4} \cdot n^{1-\alpha}) + O(n \log n \cdot \varepsilon^{-5});$$
$$S(n) \leq \max_{1/3 \leq \alpha \leq 2/3} S(n^\alpha) \cdot O(\varepsilon^{-4} \cdot n^{1-\alpha}) + S(\alpha n) \cdot O(\varepsilon^{-4} \cdot n^{1-\alpha}) + O(\varepsilon^{-1} n).$$

In the base case, there are $O(\varepsilon^{-4}N)$ graph of $O(\varepsilon^{-4})$ vertices each, so we need to spend $O(\varepsilon^{-4}N)$ time. For $N = n$, we have $T(N) = O(N \log N \log^* N) + \varepsilon^{-5}$ and $S(N) = O(\varepsilon^{-1}N \log N)$, which are the total preprocessing time and space requirements, respectively, of our oracle of additive stretch.

Theorem 5.12. (Oracle of additive stretch) *Given a non-negatively weighted, undirected planar graph of n vertices and of diameter Δ, we can construct for it an oracle of $O(\varepsilon\Delta)$ additive stretch with $O(n \log^2 n \log\log n) + \varepsilon^{-5}$ preprocessing time, $O(\varepsilon^{-1} n \log^2 n)$ space, and $O(\log(1/\varepsilon))$ query time.*

5.3.2 Approximate distance oracles

Now we show how to use our oracle of additive stretch as a building block to construct a $(1+\varepsilon)$-approximate distance oracle by using an existing technique based on *sparse neighborhood covers*. Recall that from Section 2.7 we have the following lemma.

Lemma 5.13. (Sparse neighborhood covers in planar graphs) *Given a planar graph $G = (V, E)$ of n vertices and an integer r, we can construct a collection of subsets V_i of V in $O(n \log n)$ time, such that (i) the diameter of the subgraph of $\mathcal{G}$ induced by each V_i is at most $24r + 18$, (ii) every vertex resides in $O(1)$ subsets V_i, and (iii) for every vertex v, the set of all vertices at distance at most r from v is contained in at least one of the V_i's.*

Our approach is similar to those of Kawarabayashi et al. [KST13] and of Gu and Xu [GX15]. Specifically, we first assume without loss of generality that each edge of $\mathcal{G}$ has weight at least one That can be ensured by dividing every edge weight with the minimum. Then, for every scale $r = 2^0, 2^1, \ldots, 2^{\log(\Delta)}$, we consider the graph $\mathcal{G}^{(r)}$ that ensues after deleting the edges of $\mathcal{G}$ of weight at least $24r$ and contracting those of weight at most

$r\{N^2$. Thus, each edge appears in the graphs of $O\,p\log N\,q$ scales, which we can identify in that much time. Let R be the set that contains each scale r such that $\mathcal{G}^{prq}$ has at least one edge. For each r P R, we construct $\mathcal{G}^{prq}$ and the sparse neighborhood cover of Lemma 5.13 for it, hence obtaining a collection of subsets V_i^{prq}. Then, for the induced graph of $\mathcal{G}^{prq}$ of each such subset, we build our distance oracle of $O\,p"rq$ additive stretch of Theorem 5.12. Finally, we build the 2-approximate distance oracle of $O\,pN\log N\,q$ space and $O\,p1q$ query time of Thorup [Tho04a] in $O\ \ N\log^3 N$ time. We assume that $"$ ¥ $2\{N$ because otherwise we can just run a linear-time SSSP algorithm [HKRS97].

Given two vertices of u and v, we obtain a 2-approximation d of the u-to-v distance by querying the oracle of Thorup [Tho04a]. Then, we compute the most significant bit [FW93] (this is where we need the assumption of the Word RAM) of d to identify a scale r such that $r\{2$ ¤ d ¤ r in constant time. Finally, we visit the oracle of each of the $O\,p1q$ subsets V_j^{prq} that contain u, compute an approximation d_j' of additive stretch of the u-to-v distance therein, and set d' to be the minimum of the values d_j' $"r\{2$.

Approximation factor. We now show that for any two vertices $u;v$ of $\mathcal{G}$, our oracle returns a value d' with $dist_G ru;vs$ ¤ d' ¤ p1 $O\,p"qqdist_G ru;vs$.

We first claim that r P R. To see this, recall that in $\mathcal{G}^{prq}$ all edges of weight at least $24r$ have been deleted and those of weight at most $r\{n^2$ have been contracted. Notice that the former edges do not participate in the u-to-v shortest path in $\mathcal{G}$ because the length of that path is at most r. Since the diameter of $\mathcal{G}^{prq}$ is at most $24r$ 18, these edges are not used in any shortest path therein. Let L be the largest summation of distances in any contracted path of $\mathcal{G}^{prq}$. Since $"$ ¡ $2\{N$, we have L ¤ $r\{N$ ¤ $"r\{2$. The length of the u-to-v shortest path in $\mathcal{G}$ is at least $r\{2$, so not all of its edges are contracted in $\mathcal{G}^{prq}$, i.e., r P R. This also shows that $dist_{G^{prq}} ru;vs$ ¤ $dist_G ru;vs$ ¤ $dist_{G^{prq}} ru;vs$ $"r\{2$.

From the first property of sparse neighborhood covers, we know that there is at least one subset V_i^{prq} that contains both u and v, such that the induced subgraph of $\mathcal{G}^{prq}$ has diameter at most $24r$ 18. From the third, we have $dist_{G_i^{prq}} ru;vs$ $dist_{G^{prq}} ru;vs$. Moreover, for the u-to-v distance d_i' returned by our oracle of additive stretch for $\mathcal{G}_i^{prq}$, we have $dist_{G_i^{prq}} ru;vs$ ¤ d_i' ¤ $dist_{G_i^{prq}} ru;vs$ $O\,p"rq$. Combining everything, $dist_G ru;vs$ ¤ d' $"r\{2$ ¤ $dist_G ru;vs$ $O\,p"dist_G ru;vsq$ ¤ p1 $O\,p"qqdist_G ru;vs$.

By adjusting $"$ by a constant factor, the approximation factor of our oracle can become 1 $"$.

Time and space analysis. The preprocessing time is dominated by the time required to build the oracles of additive stretch for the graphs $\mathcal{G}_i^{prq}$ associated with the sparse neighborhood cover of $\mathcal{G}^{prq}$ for each scale r P R (the same applies for the space). Each such oracle can be constructed in $O\ \ n\ \log^2 n\,p\log n$ p1{$"q^5q$ time, where n is the number of vertices

of the corresponding graph. Since each edge of $\mathcal{G}$ appears in $O(\log N)$ graphs $\mathcal{G}_i^{prq}$, the summation of n's over all oracles of additive stretch is $O(N \log N)$. Therefore, the total preprocessing time and space of our $(1+\varepsilon)$-approximate oracle is $O\left(N \log^2 N \log N \left(1/\varepsilon^5\right)\right)$ and $O\left(\left(1/\varepsilon\right) N \log^2 N\right)$ respectively. The query time is $O(\log(1/\varepsilon))$ because we spend $O(1)$ time to find the appropriate scale and query $O(1)$ oracles of additive stretch, each in $O(\log(1/\varepsilon))$ time.

New Result 6. (Approximate distance oracle) *We can construct a $(1+\varepsilon)$-approximate distance oracle for a non-negatively-weighted, undirected planar graph of n vertices with $O\left(n \log^2 n \log n \left(1/\varepsilon^5\right)\right)$ preprocessing time, $O\left(\left(1/\varepsilon\right) n \log^2 n\right)$ space, and $O(\log(1/\varepsilon))$ query time in the Word RAM.*

Chapter 6

Approximate shortest paths and distance oracles in weighted unit-disk graphs

We consider in this chapter $(1+\varepsilon)$-approximate shortest path problems in weighted unit-disk graphs. We develop the first near-linear time $(1+\varepsilon)$-approximation algorithm for the *diameter* of weighted unit-disk graphs of n vertices, running in $O(n \log^2 n)$ time for any constant $\varepsilon > 0$. Hence, we considerably improve upon the near-$O(n^{3/2})$-time algorithm of Gao and Zhang [GZ05]. Using similar ideas, we develop a $(1+\varepsilon)$-approximate *distance oracle* of $O(1/\varepsilon)$ query time for these graphs with a likewise improvement in the preprocessing time, specifically from near $O(n^{3/2})$ to $O(n \log^3 n)$. We also obtain similar new results for a number of related problems in the weighted unit-disk graph metric, such as the radius and the bichromatic closest pair.

As a further application, we employ our new distance oracle, along with additional ideas, to solve the $(1+\varepsilon)$-approximate *all-pairs bounded-leg shortest paths* (apBLSP) problem for a set of n planar points with $O(n^2 \log n)$ space, $O(\log \log n)$ query, and near $O(n^{2.579})$ preprocessing time, for any constant $\varepsilon > 0$, improving thus the near-cubic preprocessing time bound by Roditty and Segal [RS11].

The results of this chapter will be presented in SoCG 2018 [CS18].

Definitions. Let $G = (V, E)$ be graph. For any $u, v \in V$, we denote a u-to-v shortest path in G by $\pi_G(u, v)$ and its length by $dist_G(u, v)$. We also refer to $dist_G(u, v)$ as shortest-path distance or simply distance. Let $pred_G(u, v)$ be v's predecessor on $\pi_G(u, v)$. The shortest-path tree of each $u \in V$ is a spanning tree of G rooted at u, such that the u-to-v shortest-path distance for each $v \in V$ in that tree corresponds to $dist_G(s, t)$.

As mentioned in previous chapters, a weighted unit-disk graph G is defined as the intersection graph of a set of n unit-diameter disks in the plane. That is, vertices correspond

to a set S of planar points (the centers of the disks), and there is an edge between every two points of S at Euclidean distance at most one (of weight equal to that distance). We assume that G is represented implicitly by S, so only $O(n)$ space is required to store it. We are interested in the following fundamental shortest-path problems in G.

- Computing a $(1-\varepsilon)$-approximation of various parameters of G, such as the *diameter* (i.e., $\max_{s,t \in S} dist_G(s,t)$), the *radius* (i.e., $\min_{s \in S} \max_{t \in S} dist_G(s,t)$), the *bichromatic closest pair distance* of two subsets $A, B \subseteq S$ (i.e., $\min_{a \in A, b \in B} dist_G(a,b)$), et cetera.
- Designing *approximate distance oracles*, i.e., data structures that support the following query: given any $s, t \in V$, compute a value d' with $dist_G(s,t) \le d' \le (1+\varepsilon) dist_G(s,t)$ and the predecessor of t in an s-to-t path of length d'.

Finally, we study the *all-pairs bounded-leg shortest paths* (apBSLP) problem. Given a set S of n planar points, we define $G_{\le L}$ to be the subgraph of the complete Euclidean graph of S that contains only edges of weight at most L. Then, we want to preprocess S, such that given two points $s, t \in S$ and any positive number L, we can quickly compute a $(1+\varepsilon)$-approximation of the length of the s-to-t shortest path in $G_{\le L}$ (i.e., the shortest path under the restriction that each leg of the trip has length bounded by L). To see the connection of apBLSP with the earlier problems, note that $G_{\le L}$ for each fixed L is a weighted unit-disk graph, after rescaling the radii. One important difference however is that L is not fixed in apBLSP, and we want to answer queries for any of the $\binom{n}{2}$ combinatorially different L's.

Background and overview of techniques. Now, we discuss the technical challenges that arise when trying to address $(1+\varepsilon)$-approximate shortest-path problems in weighted unit-disk graphs. For now, we assume that $\varepsilon > 0$ is a constant.

Planar graph techniques. In weighted planar graphs, there are many results on developing $(1+\varepsilon)$-approximate distance oracles and $(1+\varepsilon)$-approximate diameter algorithms as we have discussed in Chapter 5. For the oracles problem, the data structures of Thorup [Tho04a] require $\tilde{O}(n)$ preprocessing time and space and $O(1)$ query time, where $\tilde{O}(f(n))$ denotes $O(f(n)\log^{O(1)} n)$ (see [Kle02, KST13, GX15, CS17b] for subsequent work). For the diameter problem, Weimann and Yuster's algorithm [WY16] requires $O(n \log^4 n)$ time, which was improved to $O(n \log^2 n)$ as we have shown in Chapter 5 (we have also shown how to eliminate the exponential dependency on $1/\varepsilon$).

All the above approximation results for planar graphs rely heavily on *shortest-path separators*: a set of shortest paths with common root, such that the removal of their vertices decomposes the graph into at least two disjoint subgraphs. Unfortunately, such separators do not seem directly applicable to unit-disk graphs, and not only because the latter may be dense. Indeed, by grid rounding we can construct a sparse weighted graph $\tilde{G}$, such that it (i) approximately preserves distances in the original unit-disk graph G (e.g., see the proof of Lemma 6.2), and (ii) is "nearly planar", in the sense that each edge

intersects at most a constant number of other edges. However, even for such a graph it is not clear how to define a shortest-path separator that divides it cleanly into an inside and an outside because edges may "cross" over the separator. At least one prior paper [YXD12] worked on extending shortest-path separators to unit-disk graphs, but the construction was complicated and achieved only constant approximation factors.

Gao and Zhang's WSPD-based technique. As mentioned in Section 2.8, Gao and Zhang [GZ05], in a seminal paper, obtained the first nontrivial set of results on shortest-path problems in weighted unit-disk graphs. To do so, they adapted a familiar technique in computational geometry, namely the *well-separated pair decomposition* (WSPD), introduced by Callahan and Kosaraju [CK95] for solving proximity problems in the Euclidean (or L_p) metric. Gao and Zhang proposed a new variant of WSPDs for the weighted unit-disk graph metric and proved that any set of n planar points has a WSPD of near-linear, namely $O(n \log n)$, size under the new definition.

Consequently, Gao and Zhang obtained a $(1+\varepsilon)$-approximate distance oracle with $O(n \log n)$ size and $O(1)$ query time. Unfortunately, the preprocessing time, $O(n^{3/2} \log n)$, is quite high, and becomes the bottleneck when the technique is applied to offline problems, such as computing the diameter. However, the issue is not constructing the WSPD itself, which can be done in near-linear time, but computing the shortest-path distances of a near-linear number of vertex pairs in the "nearly planar" graph G mentioned above. That computation takes almost $n^{3/2}$ time by showing that G has a balanced separator [MTV91, EMT95] and adapting a known exact distance oracle for planar graphs [ACC+96].

Our technique. To obtain a near-linear-time $(1+\varepsilon)$-approximation diameter algorithm and $(1+\varepsilon)$-distance oracles with near-linear preprocessing time in weighted unit-disk graphs, we follow a conceptually simple approach: we just employ known shortest-path separator techniques from the planar-graphs case [Tho04a, KST13]! However, we find shortest-path separators not in the given unit-disk graph G, but in a planar $O(1)$-*spanner* H of G. Fortunately, such spanners are already known to exist in unit-disk graphs [LCW02] and were also used by Gao and Zhang [GZ05]. Specifically, we apply divide-and-conquer over the decomposition tree that results by recursively decomposing H with shortest-path separators.

Although the above plan may sound obvious in hindsight, the details are tricky to get right. For example, how could the use of an $O(1)$-spanner eventually lead to $1+\varepsilon$ approximation factor? The known divide-and-conquer approaches for planar graphs select a small number of vertices, called *portals*, along each separator and compute distances from each with a single-source shortest paths (SSSP) algorithm. That works well because a shortest path in a planar graph crosses a separator only at vertices. In our case, however, we need to use the original (non-planar, unit-disk) graph G when computing distances from portals, but therein a shortest path could "cross" the separator over an edge. We show that we can nevertheless re-route such a path to pass through a separator vertex without increasing the length by much, by using the fact that H is an $O(1)$-spanner.

6.1 Approximate diameter and distance oracles

Let S be a set of planar points whose weighted unit-disk graph G has diameter Δ. A key subproblem in both (i) computing a $(1+\varepsilon)$-approximation of the diameter of G and (ii) building a $(1+\varepsilon)$-approximate distance oracle for G is the construction of a distance oracle with *additive stretch* $O(\varepsilon\Delta)$: a data structure, such that given any $s, t \in S$, we can quickly compute a value $\hat{d}$ with $dist_G(s,t) \le \hat{d} \le dist_G(s,t) + O(\varepsilon\Delta)$. This is because we can apply such an oracle, along with existing techniques, to address the two original problems, as we show in Section 6.1.3. In Section 6.1.1 we give two preliminary ingredients and then describe our oracle of additive stretch in Section 6.1.2.

6.1.1 Preliminaries

The first ingredient we need is the existence of a planar spanner with constant stretch factor in any weighted unit-disk graph.

Lemma 6.1. (Planar spanner) *Given a set S of n planar points, we can find a subgraph H of its weighted unit-disk graph G in $O(n \log n)$ time, such that H is a (i) planar graph and (ii) c-*spanner *of G, i.e., for every $s, t \in S$, $dist_G(s,t) \le dist_H(s,t) \le c\,dist_G(s,t)$ for some constant $c > 0$.*

Li, Calinescu, and Wan [LCW02] proved the above lemma with $c = 2.42$ by simply building the Delaunay triangulation of the given points and discarding edges of weight more than one. However, the analysis of the stretch factor c is nontrivial.

The second ingredient is an efficient algorithm for the single-source shortest paths (SSSP) problem in weighted unit-disk graphs. The currently best exact result is due to Cabello and Jejčič [CJ15], requires $O(n \log^{12} \log n)$ time and employs complicated dynamic data structures for additively weighted Voronoi diagrams [Cha10a, KMR+17]. For our purposes though, it suffices to consider the $(1+\varepsilon)$-approximate version of the problem instead, i.e., given a set of points S and a source $s \in S$, compute a path of length $\hat{d}(s,t)$ for each $t \in S$, such that $dist_G(s,t) \le \hat{d}(s,t) \le (1+\varepsilon)\,dist_G(s,t)$. Our algorithm first finds a sparse graph $\mathcal{G}$ that $(1+\varepsilon)$-approximately preserves distances in G, i.e., for any $s, t \in S$, there are vertices c_s, c_t of $\mathcal{G}$, such that $dist_G(s,t) \le dist_{\mathcal{G}}(c_s, c_t) \le (1+\varepsilon)\,dist_G(s,t)$). Then, it runs Dijkstra's algorithm in $\mathcal{G}$. Sparsification in weighted unit-disk graphs has been used before (e.g., see [GZ05, Section 4.2]).

Lemma 6.2. (Approximate SSSP) *Given a set S of n planar points, we can solve the $(1+\varepsilon)$-approximate SSSP problem in its weighted unit-disk graph G in $O((1+\varepsilon)\{\varepsilon^2 n \log n)$ time.*

Proof. First, we build a uniform grid of side length ε and construct a sparse weighted graph $\mathcal{G}$ by placing a vertex at each non-empty grid cell and an edge between every two

such cells c and c' iff there exist points $p \in c$ and $p' \in c'$ with $\|p - p'\| \leq 1$. The weight of that edge is equal to the maximum Euclidean distance of c and c'. Each grid cell has at most $O(1/\varepsilon^2)$ neighbors, so $\mathcal{B}$ has at most $O(1/\varepsilon^2 n)$ edges and can be constructed in $O(1/\varepsilon^2 n \log n)$ time with a Euclidean bichromatic closest pair algorithm [PS85] over $O(1/\varepsilon^2 n)$ pairs of grid cells.

Let s and t be two points of S. If $\|s - t\| \leq 1$, we can trivially return $\|s - t\|$. Otherwise, let $p_0 p_1 \cdots p_\ell$, with $p_0 = s$ and $p_\ell = t$, be the shortest path in G from s to t. Two consecutive edges therein have lengths whose sum is at least one because otherwise we could take a short-cut and obtain a shorter path. Thus, $dist_G(s,t) \geq \ell/2$. Consider the path $c_0 c_1 \cdots c_\ell$ in $\mathcal{B}$, where each c_i is the cell that contains p_i. Since, for each c_i, c_{i+1}, $\|p_i - p_{i+1}\| \leq dist_{\mathcal{B}}(c_i, c_{i+1}) \leq \|p_i - p_{i+1}\| + O(\varepsilon)$, it follows that $dist_G(s,t) \leq dist_{\mathcal{B}}(c_0, c_\ell) \leq dist_G(s,t) + O(\varepsilon) \cdot dist_G(s,t) + 1 + O(\varepsilon) \leq (1+O(\varepsilon)) dist_G(s,t) + 1$.

Given a source $s \in S$, we can invoke Dijkstra's algorithm in $\mathcal{B}$ to compute the shortest path tree from s and return $\tilde{d}(s,t) = dist_G(c_s, c_t)$ for each $t \in S$, where c_s and c_t are the grid cells that contain s and t, respectively. From the previous paragraph, we have $dist_G(s,t) \leq \tilde{d}(s,t) \leq (1+O(\varepsilon)) dist_G(s,t) + 1$. We can easily modify our algorithm to also find, for each $t \in S$, an s-to-t path in G of length $\tilde{d}(s,t)$, by appending s and t at the ends of the s-to-t shortest path in $\mathcal{B}$ and replacing each c_i and c_{i+1} with the bichromatic closest pair of $(S \cap c_i, S \cap c_{i+1})$ in G. Notice that these pairs have been found while constructing $\mathcal{B}$. □

6.1.2 Distance oracles with additive stretch

We now describe a distance oracle with additive stretch for an arbitrary weighted, undirected graph $G = (V, E)$ of n vertices and of diameter Δ that has the following properties, which are the only ones needed from weighted unit-disk graphs.

(I) There exists a planar *c-spanner* H of G, for some constant $c > 0$.

(II) For any induced subgraph of G with n' vertices, the $(1+\varepsilon)$-approximate SSSP problem can be solved in $T(n')$ time, for some function $T(\cdot)$ such that $T(n')/n'$ is nondecreasing.

(III) Every edge weight in G is at most $\varepsilon\Delta$.

If G is a weighted unit-disk graph, Lemmas 6.1 and 6.2 imply (I) and (II), respectively, where $c = 2{:}42$ and $T(n') = O(1/\varepsilon^2 n' \log n')$, and (III) holds as long as $\Delta \geq 1/\varepsilon$.

Shortest-path separators in H. Although G may not have nice shortest-path separators, we know by planarity that H does. Thus, as described in Section 2.5, we apply a known shortest-path separator decomposition for H, namely the version of Kawarabayashi, Sommer, and Thorup [KST13, Section 3.1], to compute in $O(n \log n)$ time a *decomposition tree* $\mathcal{T}$ with the following properties.

- $\mathcal{T}$ has $O(1)$ degree and $O(\log n)$ height.
- Each node x of $\mathcal{T}$ is associated with a subset $V^{x} \subseteq V$. The subsets V^{x} over all children x of t are disjoint and contained in V^{t}. If t is the root, $V^{t} = V$; if t is a leaf, V^{t} has $O(1)$ size.
- Each non-leaf node t of $\mathcal{T}$ is associated with a set of $O(1)$ paths, called *separator paths*, which are classified as "internal" and "external". The internal separator paths cover precisely all vertices of $V^{t} \setminus \bigcup_{\text{child } x \text{ of } t} V^{x}$, while the external are outside of V^{t}.
- For each child x of a non-leaf node t, every neighbor of the vertices of V^{x} in H is either in V^{x} or in one of the (internal or external) separator paths at t.
- Each separator path is a shortest path in H and, in particular, has length at most the diameter $\Delta(H)$ of H (which is at most $c\Delta$).

Our data structure. To construct an additive oracle with $O(\varepsilon\Delta)$ stretch for G, we construct the above decomposition tree $\mathcal{T}$ and augment it with extra information, as follows. Let t be an internal node of $\mathcal{T}$ and P one of its internal separator paths. Since P has length at most $\Delta(H) \leq c\Delta$, we can select, with a linear walk, a set of $O(1/\varepsilon)$ vertices thereon, called *portals*, such that each consecutive pair of them is at distance at most $\varepsilon\Delta$ on it.

Let P^{t} denote the set of all portals over all internal separator paths at a non-leaf node t of $\mathcal{T}$. For each such node and for each $p \in P^{t}$ and $v \in V^{t}$, we invoke $O(1/\varepsilon)$ times the SSSP algorithm from Property (II) to compute a $(1+\varepsilon)$-approximation, $\tilde{d}(p,v)$, of the shortest-path distance from p to v in the subgraph of G induced by V^{t}. Then, for each leaf t, we just find and store all pairwise distances in the subgraph of G that is induced by V^{t}. Overall, our oracle requires $O((1/\varepsilon)T(n) \log n)$ preprocessing time and $O((1/\varepsilon)n \log n)$ space.

Query algorithm. Given two vertices $s, t \in V$, we first identify all $O(\log n)$ nodes in $\mathcal{T}$, such that both $s \in V^{t}$ and $t \in V^{t}$. To do so, we trivially start from the root and go down the tree along a path. For each such non-leaf node t, we compute a value $\tilde{d}(s,t) = \min_{p \in P^{t}} \{\tilde{d}(s,p) + \tilde{d}(p,t)\}$ in $O(1/\varepsilon)$ time. If t is a leaf, $\tilde{d}(s,t)$ is the exact shortest-path distance in the subgraph of G induced by V^{t}, which we have already computed. Finally we return the minimum, $\tilde{d}(s,t)$, over all $\tilde{d}(s,t)$. The total query time is $O((1/\varepsilon)\log n)$.

Stretch analysis. We want to prove that for any $s, t \in V$, the value $\tilde{d}(s,t)$ that our oracle returns is such that $dist_G(s,t) \leq \tilde{d}(s,t) \leq dist_G(s,t) + O(\varepsilon\Delta)$. The left side of

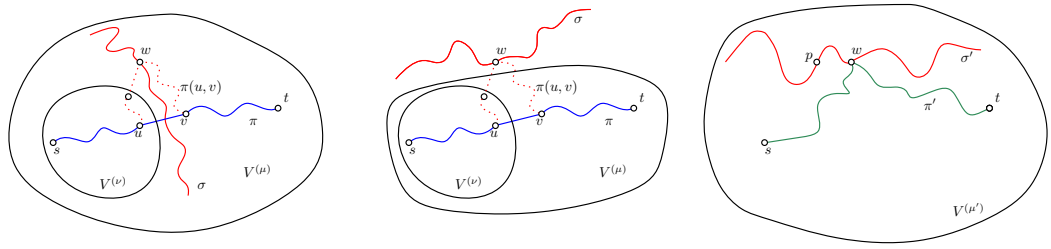


Figure 6.1: Detour through a vertex of a separator path in Claim 6.1, where may be internal, as in (a), or external, as in (b). Detour through a portal in Claim 6.2 in (c).

the inequality clearly holds because $d(r,s;t)$ corresponds to the length of an s-to-t path in a subgraph of G. To prove the right side, let be the shortest s-to-t path in G, and let be the lowest node in $\mathcal{T}$, such that all vertices of lie in $V^{p,q}$. We assume that is a non-leaf node because otherwise we have already computed $dist_G(r,s;t)$ exactly.

Although is a path in the (not necessarily planar) graph G, not H, we show that it is possible to re-route it to pass through a vertex on a separator path of without increasing its length by much.

Claim 6.1. (Detour through a separator vertex) *There exists an s-to-t path π_1 in G that (i) passes through some vertex w on a separator path of , (ii) uses only vertices of $V^{p,q}$ (except maybe for w itself) and (iii) has length at most $dist_G(r,s;t)$ $2c''\Delta$.*

Proof. We assume that none of the vertices on lie on a separator path of because otherwise we can just set π_1 . Let be the child of with $s \in V^{p,q}$, let u be the last vertex on that lies in $V^{p,q}$ (note that u t, by definition of), and let v be the next vertex after u thereon. By Property (I) of G, there is a path $\pi_{u,v}$ from u to v in H of length at most c (the weight of uv), which is at most $c''\Delta$ by Property (III). Let w be the first vertex on $\pi_{u,v}$ that lies outside of $V^{p,q}$, which exists since v is outside of $V^{p,q}$. Then, from the fourth property of $\mathcal{T}$, we know that w must be on an (internal or external) separator path of . Thus, we set π_1 to be the path that goes from s to u along , then from u to w along $\pi_{u,v}$ (which uses only vertices in $V^{p,q}$ as intermediates), then back from w to u along $\pi_{u,v}$, and finally from u to t along . See Figure 6.1(a) (where is internal) and 6.1(b) (where is external). □

Next, we note how to further re-route to pass through a portal.

Claim 6.2. (Detour through a portal) *There exists another s-to-t path π_2 in G that (i) passes through a portal p on a separator path π_1' of π_1, where π_1' is some ancestor of , (ii) uses only vertices of $V^{p,q}$, and (iii) has length at most $dist_G(r,s;t)$ $p2c$ $2q''\Delta$.*

Proof. Let w be as in Claim 6.1, and let ^{1} be the lowest ancestor of with $w \in V^{p\ ^1q}$. Notice that if $w \in V^{p\ q}$, then ^{1}. Then w must be on an internal separator path ^{1} in ^{1}, whose existence is guaranteed by the third property of $\mathcal{T}$. Let p be the portal on ^{1} that is closest to w. Thus, the p-to-w distance on ^{1} is at most $O(p''\Delta q$. We set ^{2} to be the path the goes from s to w along ^{1}, then from w to p along ^{1} and back from p to w, and, finally, from w to u along ^{1}. See Figure 6.1(c). □

Let ^{1} be as in Claim 6.2. It follows that $\lceil rs; t \rceil \leq \lceil {}_1rs; t \rceil \leq d'_1(rs; p) \quad d'_1(rp; t) \leq dist_G(rs; t) \quad O(p''\Delta q$.

Theorem 6.3. (General distance oracles of additive stretch) *Given a weighted graph G of n vertices and of diameter Δ that satisfies Properties (I)–(III), along with the subgraph H from Property (I), we can construct for it a distance oracle of $O(p''\Delta q$ additive stretch, $O(pp1\{''qTpnq \log nq$ preprocessing time, $O(pp1\{''qn \log nq$ space, and $O(pp1\{''q \log nq$ query time.*

As we explained earlier, weighted unit-disk graphs satisfy Properties (I)-(III) of G, so we have the following theorem.

Corollary 6.4. (Distance oracles of additive stretch in unit-disk graphs) *Given a set S of n planar points, such that the weighted unit-disk graph of S has diameter $\Delta \geq 1\{''$, we can construct for that graph a distance oracle of $O(p''\Delta q$ additive stretch, $O\ p1\{''q^3n \log^2 n$ preprocessing time, $O(pp1\{''qn \log nq$ space, and $O(pp1\{''q \log nq$ query time.*

6.1.3 Applications

We now describe how to employ Corollary 6.4 to compute a $(p1\ \ ''q$-approximation of the diameter of a unit-disk graph and how to build a $(p1\ \ ''q$-approximate distance oracle for it. Let S be a set of planar points, let G be the weighted unit-disk graph defined by S, and let H be an $O(p1q$-planar spanner of G.

Approximate diameter. Recall that as explained in Section 2.8, to approximate the diameter of G, we can find a set of $O(pp1\{''q^4n \log nq$ pairs of points of S in $O(pp1\{''q^4n \log nq$ time, such that the shortest-path distance between any two vertices in G can be $(p1\ \ ''q$-approximated by the shortest-path distance between one of these pairs, which can be found in $O(p1q$ time.

First, we compute in $O(pn \log nq$ time [PS85] the Euclidean diameter Δ_0 of S. If $\Delta_0 \geq 1\{''$, then $\Delta \geq 1\{''$, and, to compute a $(p1\ \ ''q$-approximation of Δ, we can query the oracle of Corollary 6.4 of $O(p''\Delta q$ additive stretch with all $O(pp1\{''q^4n \log nq$ pairs of Lemma 2.4 and return the maximum. Thus the approximation factor is $1\ \ O(p''q$. The total time required for this case is $O(ppp1\{''q^4n \log n\ \ p1\{''q \log nq$.

If $1 \le \Delta_0 \le 1+\varepsilon$, the problem is more straightforward because we can construct the sparsified graph H from the proof of Lemma 6.2, which preserves distances approximately, and then run a standard all-pairs shortest paths (APSP) algorithm therein. Since H has $n' = O(n\Delta_0/\varepsilon^2) = O(n/\varepsilon^2)$ vertices and $m' = O(n/\varepsilon^2)\cdot O(1/\varepsilon^2) = O(n/\varepsilon^6)$ edges, we need $O(n'^2 \log n' + n'm') = O(n/\varepsilon^{10})$ time for this case. Finally, if $\Delta_0 \ge 1$, the unit-disk graph is a complete Euclidean graph, so we just return Δ_0.

New Result 7. (Approximate diameter) *Given a set S of n planar points, we can compute in $O(n/\varepsilon^5 + n\log^2 n) \le O(n/\varepsilon^{10})$ time a $(1+\varepsilon)$-approximation of the diameter of its weighted unit-disk graph.*

Remark: It is probably possible to avoid WSPDs by combining our techniques with those of Weimann and Yuster for planar graphs [WY16]. However, the ε dependency would increase to $2^{O(1/\varepsilon)}$.

Approximate distance oracles. To build a distance oracle of $(1+\varepsilon)$-approximation factor for a weighted unit-disk graph, we employ the oracle of Corollary 6.4 of $O(\varepsilon\Delta)$ additive stretch as a building block using a known technique, called *sparse neighborhood covers*, as described in Section 2.8 and also used in Section 5.3. We restate that lemma as follows.

Lemma 6.5. (Sparse neighborhood cover) *Given a weighted planar graph H of n vertices and a value r, we can construct, in $O(n\log n)$ time, a collection of subsets V_i of V, such that (i) the diameter of the subgraph of H induced by each V_i is $O(r/\varepsilon)$, (ii) every vertex resides in $O(1)$ subsets, and (iii) for every vertex v, the set of all vertices at distance at most r from v in H is contained in at least one of the V_i's.*

Every shortest-path distance in G is upper bounded by n, so we first apply the above lemma to H for each value of $r \in \{2^0, 2^1, \ldots, 2^{\log n}\}$. Thus, we obtain collections of subsets $V_i^{(r/\varepsilon)}$ and then build the distance oracle of Corollary 6.4 for the weighted unit-disk graph of each $V_i^{(r/\varepsilon)}$. The total preprocessing time and space over all $O(\log n)$ choices of r is $O(\log n \cdot n/\varepsilon^3 \log^2 n)$ and $O(\log n \cdot n/\varepsilon \log n)$ respectively. Given $s, t \in S$, we consider each r and each subset $V_i^{(r/\varepsilon)}$ that contains both s and t, query the oracle for $V_i^{(r/\varepsilon)}$, and return the minimum. The total query time over all $O(\log n)$ choices of r and $O(1)$ choices of $V_i^{(r/\varepsilon)}$ (Lemma 6.5(ii)) is $O(\log n \cdot 1/\varepsilon \log n)$.

If $dist_G(s,t) \ge 1+\varepsilon$, let $r \ge c/\varepsilon$ be such that $dist_G(s,t) \in (2c/r, c/r]$. Then, each vertex on the shortest path from s to t in G is at distance at most $cdist_G(s,t) \le r$ from s in H, so it is contained in a common subset $V_i^{(r/\varepsilon)}$. Hence, we approximate $dist_G(s,t)$ with an additive error of $O(\varepsilon r) \le O(\varepsilon\, dist_G(s,t))$, obtaining thus $1+O(\varepsilon)$ approximation factor.

If $1 \le dist_G(s,t) \le 1+\varepsilon$, we simply build the sparsified graph H from the proof of Lemma 6.2, which preserves distances approximately. Then, from every vertex, we pre-compute the distances to all grid cells at Euclidean distance at most $1+\varepsilon$ by running

Dijkstra's algorithm on a subgraph of G with n^{1} $O\,p1\{"q^4q$ vertices and $O\,pp1\{"q^2n^{1}q$ $O\,pp1\{"q^6q$ edges in $O\,pp1\{"q^6\log p1\{"qq$ time. The total preprocessing time and space over all sources is $O\,pp1\{"q^6n\log p1\{"qq$ and $O\,pp1\{"q^4nq$ respectively. Finally, if $dist_G rs;ts \neq 1$, the shortest-path distance of s and t is their Euclidean distance. We do not know a priori which of the cases we are in, so we try all of them and return the minimum distance found.

New Result 8. (Approximate distance oracles in weighted unit-disk graphs) *Given a set S of n planar points, we can construct a $p1$ $"q$-approximate distance oracle for its weighted unit-disk graph with O $p1\{"q^5n\log^3 n$ $p1\{"q^6n\log p1\{"q$ preprocessing time, $O\,pp1\{"q^4n\log nq$ space, and $Op1q$ query time.*

To reduce the query time, we can combine the above method with Gao and Zhang's WSPD-based oracle [GZ05, Section 5.1], which requires $O\,p1q$ query time and $O\,pp1\{"qn\log nq$ space. Its construction time is dominated by finding $p1$ $"q$-approximate shortest-path distances for $O\,pp1\{"q^4n\log nq$ pairs, but we can compute these distances by querying our oracle of Theorem 8 in O $p1\{"q^4n\log n$ $p1\{"q\log^2 n$ total time.

Corollary 6.6. (Approximate distance oracle with $O\,p1q$ query time) *Given a set S of n planar points, we can construct a $p1$ $"q$-approximate distance oracle for its weighted unit-disk graph with O $p1\{"q^5n\log^3 n$ $p1\{"q^6n\log p1\{"q$ preprocessing time, $O\,pp1\{"q^4n\log nq$ space, and $O\,p1q$ query time.*

Similarly, we can use the distance oracle of Theorem 8 to improve Gao and Zhang's results for other distance-related problems on weighted unit-disk graphs:

Corollary 6.7. (Approximate radius and bichromatic closest pair) *Given a set S of n planar points, we can compute a $p1$ $"q$-approximation of the radius of the weighted unit-disk graph G of S or of the bichromatic closest pair distance of two given subsets $A;B \subseteq S$ in G in O $p1\{"q^5n\log^3 n$ $p1\{"q^6n\log p1\{"q$ time.*

Remarks:

- For the sake of simplicity, we did not optimize the $poly p1\{";\log nq$ factors.
- Our distance oracle in Theorem 8 can be easily modified to report an approximate shortest path, not just its distance, in additional time proportional to the number of edges in the path. To do so, every time we find approximate shortest distances in a subgraph from a portal, we also store its approximate shortest-path tree.
- The same approach gives $p1$ $Op"qq$-approximation results for *unweighted* unit-disk graphs, assuming that the diameter and the distances of the query vertices exceed $\Omega p1\{"q$. Specifically, Lemma 2.4 can be modified for the unweighted case, but the error now has an extra additive term of 4 $Op"q$ [GZ05, Lemma 6.2], which can be ignored under our assumption. Also, we need to replace the SSSP algorithm of Lemma 6.2 with the $O\,pn\log nq$-time exact SSSP algorithm by Cabello and Jejčič [CJ15] or by Chan and Skrepetos [CS16].

6.2 Approximate apBLSP

In this section, we study the $(1+\varepsilon)$-approximate apBSLP problem. Given a set S of n planar points, let G be its complete weighted Euclidean graph, let $w_1, w_2, \ldots, w_N$, where $N = \binom{n}{2}$, be the weights of the edges of G in non-decreasing order, and let G^i be the subgraph of G that contains only the edges of weight at most w_i. We can assume that $w_1 \geq 1$. Otherwise, we can impose that assumption by simply translating and rescaling S in linear time. We want to preprocess S into a data structure, such that we can quickly answer $(1+\varepsilon)$-approximate *bounded-leg distance* queries, i.e., given $s, t \in S$ and a positive number L, compute a $(1+\varepsilon)$-approximation of $dist_{G^i}(s,t)$, where i is the largest integer with $w_i \leq L$. First, we briefly review the previous methods of Bose et al. [BMN 04] and of Roditty and Segal [RS11], in Section 6.2.1, and then describe our own approach, in Section 6.2.2.

6.2.1 Previous methods

Let $s, t \in S$, and let $(c_{ps,tq})$ be the minimum index, such that s and t are connected in $G^{c_{ps,tq}}$. Since each G^i is a subgraph of G^{i+1}, we have $dist_G(s,t) \leq dist_{G^N}(s,t) \leq \cdots \leq dist_{G^{c_{ps,tq}}}(s,t)$. Moreover, the s-to-t shortest path in any G^i with $i < c_{ps,tq}$ must have an edge of weight at least $w_{c_{ps,tq}}$, so $dist_G(s,t) \geq w_{c_{ps,tq}}$. Any shortest path has at most $n-1$ edges, thus $dist_{G^{c_{ps,tq}}}(s,t) \leq (n-1)dist_G(s,t)$. Therefore, as Roditty and Segal [RS11, Section 2] noticed, we can compute and store a $(1+\varepsilon)$-approximation of the s-to-t shortest-path distance for each $s, t \in S$ in only $O(\log_{1+\varepsilon} n)$ graphs, such that a bounded-leg distance query can be answered with a binary search in $O(\log\log_{1+\varepsilon} n)$ time.

Specifically, for every $s, t \in S$ and $j \in \{0, 1, \ldots, \lceil\log_{1+\varepsilon} n\rceil\}$, let $I^j(s,t)$ be the set of indices of the graphs G^i, such that $(1+\varepsilon)^j dist_G(s,t) \leq dist_{G^i}(s,t) \leq (1+\varepsilon)^{j+1} dist_G(s,t)$. If $I^j(s,t) \neq \emptyset$, we create two values $m^j(s,t)$ and $\ell^j(s,t)$, where the former is any index therein and the latter is equal to $w_{m^j(s,t)}$. Else, $m^j(s,t)$ and $\ell^j(s,t)$ are undefined. The total space required over all pairs of S is $O(n^2 \log_{1+\varepsilon} n)$. Then, given a positive number L, we can find the largest i among the $m^j(s,t)$'s such that $w_i \leq L$ with a binary search over the $\ell^j(s,t)$'s in $O(\log\log_{1+\varepsilon} n)$ time and return a $(1+\varepsilon)$-approximation of the s-to-t shortest-path distance in G^i.

To compute a possible index for $m^j(s,t)$ for every $s, t \in S$ and $j \in \{0, 1, \ldots, \lceil\log_{1+\varepsilon} n\rceil\}$, Roditty and Segal performed $O(n^2 \log_{1+\varepsilon} n)$ independent binary searches, each making $O(\log n)$ $(1+\varepsilon)$-approximate bounded-leg distance queries (i.e., a query to find a $(1+\varepsilon)$-approximation of the s-to-t shortest-path distance in some graph G^i). Instead, we group the queries for all s, t, j into $O(\log n \log_{1+\varepsilon} n)$ rounds of n^2 offline queries each, where "offline" means that the queries in every round are given in advance.

Lemma 1. (Framework for approximate apBLSP) *Given a set S of n planar points, we can construct a data structure for the $(1+\varepsilon)$-approximate apBLSP problem of $O((1+\varepsilon)^2 n^2 \log n)$*

space, $O(\log \log n \cdot \log(1/\varepsilon))$ *query, and* $O(T_{\text{oine}}(n, n^2, 1/\varepsilon) \cdot \varepsilon^{-1} \cdot \log(1/\varepsilon))$ *preprocessing time, where* $T_{\text{oine}}(n', q', 1/\varepsilon')$ *denotes the total time for answering* q' *offline* $(1+\varepsilon')$*-approximate bounded-leg distance queries for an* n'*-point set.*

Naively we could construct in near linear time a sparse $(1+\varepsilon)$-spanner of every graph G^i and then run Dijkstra's algorithm therein to answer each query (a similar idea was used by Roditty and Segal). Thus a near-cubic bound would be obtained for $T_{\text{oine}}(n, n^2, 1/\varepsilon)$. Instead, we show that by employing our $(1+\varepsilon)$-approximate distance oracle of Corollary 6.6 for weighted unit-disk graphs as a subroutine, we can obtain a truly subcubic bound on $T_{\text{oine}}(n, n^2, 1/\varepsilon)$, as we next describe.

6.2.2 Improved method

To obtain our improved method, we view the problem of answering for each $s, t \in S$ and $j \in \{0, 1, \ldots, \lceil \log_{1+\varepsilon} n\sqrt{n^2} \rceil\}$ approximate offline bounded-leg distance queries as the problem of constructing and querying the following offline *semi-dynamic* (actually insertion-only) distance oracle.

Subproblem 1. (Semi-dynamic approximate distance oracles) *Given an arbitrary graph of* n *vertices with edge weights in* $[1, 8n]$*, we want to perform an offline sequence of* q *operations, each of which is either an edge insertion, or a query to compute a* $(1+\varepsilon)$*-approximation of the shortest-path distance between two vertices. Let* $T_{\text{dyn}}(n, q, 1/\varepsilon)$ *be the complexity of this problem.*

We could reduce our problem to Subproblem 1 by naively inserting the $O(n^2)$ edges of G in increasing order of weight to an initially empty graph and mix that sequence of insertions with the given sequence of bounded-leg distance queries. Hence, we would have that $T_{\text{oine}}(n, n^2, 1/\varepsilon) = O(T_{\text{dyn}}(n, n^2, 1/\varepsilon))$.

We propose a better reduction that employs a simple periodic rebuilding trick. First, we divide the sequence of the q edge insertions and queries into $O(q/r)$ *phases* of at most r operations each, where r is a parameter to be set later. At the beginning of each phase, the current graph is a weighted unit-disk graph (after rescaling), so we can build the $(1+\varepsilon)$-approximate distance oracle of Corollary 6.6 in $O((1/\varepsilon^5) n \log^3 n)$ time. Then we query that oracle in $O(r^2)$ total time to approximate the shortest-path distances between all pairs of vertices that are involved in the upcoming r operations (i.e., are endpoints of the edges to be inserted, or belong to the pairs to be queried). We build the complete graph over these at most $2r$ vertices with the approximate shortest-path distances as edge weights. Each phase can then be handled by r edge insertions/queries on this smaller graph in $O(T_{\text{dyn}}(2r, r, 1/\varepsilon))$ time. The resulting approximation factor is at most $(1+\varepsilon)^2 = 1 + \Theta(\varepsilon)$. Thus, for $q = n^2$ we get the following bound:

$$T_{\text{oine}}(n, n^2, 1/\varepsilon) = \Theta(\varepsilon^{-1}) \cdot O\left(\frac{n^2}{r}\right) \cdot (1/\varepsilon^5) n \log^3 n + r^2 \cdot T_{\text{dyn}}(2r, r, 1/\varepsilon) \quad (6.1)$$

To solve Subproblem 1, we could do nothing during insertions and in each query rerun Dijkstra's algorithm from scratch, thus obtaining $T_{\text{dyn}}(2r; r; 1 - \varepsilon) = O(r^3)$. Then by setting $r = (1/\varepsilon)^{3} n^{1/3} \log n$, we can obtain a still better bound $T_{\text{one}}(n; n^2; 1 - \varepsilon) = O((1/\varepsilon)^{10/3} n^{8/3} \log^2 n)$, which is truly subcubic.

Actually, by using fast matrix multiplication and additional techniques, we can establish a better bound on $T_{\text{one}}(n; n^2; 1 - \varepsilon) = O(1/\varepsilon)$. Our idea is to recursively divide phases into subphases, as in the proof of the following lemma. (Note that this lemma actually holds for general graphs. Although (semi-)dynamic shortest paths have been extensively studied in the literature, we are unable to find this particular result.)

Lemma 6.8. (A semi-dynamic approximate distance oracle) *We can solve Subproblem 1 in $T_{\text{dyn}}(2r; r; 1 - \varepsilon) = O(1/\varepsilon)$ total time, where ω is the matrix-multiplication exponent and $\overline{W}$ is an upper bound on the maximum (finite) shortest-path distances.*

Proof. Let H be the input graph of $2r$ vertices, and let H^1 be the graph that results from performing to H all edge insertions of the first $r-2$ operations. We run the $O((1/\varepsilon) r^{\omega} \log \overline{W})$-time $(1 - \varepsilon)$-approximate APSP algorithm of Zwick [Zwi02] on H and H^1 and answer all distance queries therein. Then, we construct two graphs H_1 and H_2 of r vertices each, where H_1 (respectively H_2) is the complete graph over all vertices that are involved in the first (respectively last) $r-2$ operations. We set each edge weight in H_1 (respectively H_2) to be a $(1 - \varepsilon)$-approximation of the shortest-path distance of its endpoints in H (respectively H^1) (which we have already computed), increasing thus the error by a $1 - \varepsilon$ factor. Finally, we apply recursion in H_1 and H_2.

The running time of our approach is $T_{\text{dyn}}(2r; r; 1 - \varepsilon^i) \leq 2T_{\text{dyn}}(r; r-2; 1 - \varepsilon^{i-1}) + O((1/\varepsilon) r^{\omega} \log \overline{W})$, where initially $i = 1$. Thus, $T_{\text{dyn}}(2r; r; 1 - \varepsilon) = O((1/\varepsilon) r^{\omega} \log r \log \overline{W})$. The approximation factor is $(1 - \varepsilon)^{\log r} \geq 1 - \Theta(\varepsilon \log r)$, which can be refined to $1 - \Theta(\varepsilon)$, by resetting $\varepsilon \leftarrow \varepsilon / \log r$. □

Combining (6.1) with the above lemma gives

$$T_{\text{one}}(n; n^2; 1 - \varepsilon) = O\left(\frac{n^2}{r}\right) + (1/\varepsilon) r^{\omega} \log^3 n + (1/\varepsilon) r^{\omega} \log r \log \overline{W}.$$

Setting $r = n^{1/(\omega+1)}$ yields $T_{\text{one}}(n; n^2; 1 - \varepsilon) = O((1/\varepsilon) n^{2\omega/(\omega+1)} \log^3(n\overline{W}))$, where $\overline{W} = nW$, and W is the spread of S, i.e., the ratio of the maximum-to-minimum pairwise Euclidean distance.

New Result 9. (Approximate all-pairs bounded leg shortest paths) *Given a set S of n planar points of spread W, we can construct a data structure for the $(1 - \varepsilon)$-approximate ap-BLSP problem with $O((1/\varepsilon) n^{2\omega/(\omega+1)} \log^5(nW))$ preprocessing time, $O((1/\varepsilon) n^2 \log n)$ space, and $O(\log \log n + \log(1/\varepsilon))$ query time.*

The current best bound on the matrix-multiplication exponent [Vas12, Le 14] is $!$ 2:373, which gives a preprocessing time of O p1{"q$^6 n^{2:579}$ log^{5}pnWq .

Remark: For the sake of simplicity, we did not optimize the polyp1{"; logpnWqq factors. It might be possible to avoid the dependency on the spread W by using known techniques, such as balanced quadtrees.

Chapter 7

Open problems

Although many shortest-path problems in geometric intersection graphs were studied in this thesis, quite a few remain open. Next, we discuss some of them. Let n be the number of vertices of each graph under consideration.

A big open problem is that of computing the diameter of a planar graph exactly in near linear time. The recent algorithms of Cabello [Cab17a] and of Gawrychowski et al. [GMWWN18], both employing Voronoi diagrams, require $\tilde{O}(n^{11/6})$ and $\tilde{O}(n^{5/3})$ time respectively, where $\tilde{O}(f(n))$ denotes $O(f(n)\log^{O(1)} n)$.

Open Problem 1. *Compute the diameter of a planar graph in near linear time.*

A similar problem concerns the exact computation of the diameter of a weighted unit-disk graph. The fastest known way to do so is to employ the SSSP algorithm of Cabello and Jejčič [CJ15] from each vertex of the graph in nearly $O(n^2 \log^{12} n)$ total time and return the maximum distance found. Thus, developing an exact truly-subquadratic-time algorithm is an intriguing challenge.

Open Problem 2. *Compute the diameter of a weighted unit-disk graph in truly subquadratic time.*

All results in this thesis concerned *static* shortest-path problems (i.e., the input never changes), but sometimes it is necessary to study the *dynamic* version of these problems, where edges and/or vertices may be inserted and/or deleted. For example, in a country road network, a road may have to be closed for a time to undergo construction. In unit-disk graphs no such results are known, while in planar graphs dynamic distance oracles have been studied both in the exact [KMNS12] and the approximate setting [KS98, ACG12, ACD+16]. However, the query and update times therein are polynomials of the input size or of the maximum edge weight.

Open Problem 3. *Construct a dynamic distance oracle for a planar or unit-disk graph with polylogarithmic query and update times.*

Another important problem, stemming from ad-hoc communication networks, is *routing*. Therein, we want to assign a *label* and a *routing table* to each node of a graph, such that a *routing scheme* can use them to transmit a message from a source to a destination node quickly. Sometimes, the routing scheme also needs a *header*. Ideally, the labels, the routing tables, and the headers are composed of only a few bits, and the routing scheme quickly devises a path whose length is close to optimal. In non-negatively-weighted, directed planar graphs, Thorup's [Tho04a] $(1+\varepsilon)$-approximate routing scheme uses labels, routing tables, and headers of $O(\log n)$ bits. However, in weighted unit-disk graphs, the best $(1+\varepsilon)$-approximate routing scheme, due to Kaplan et al. [KMRS18], requires labels, routing tables, and headers of $O(\log n)$, $O\left((1/\varepsilon)^5 \log^2 n \log^2 \Delta\right)$, and $O(\log n \log \Delta)$ bits respectively, where Δ is the diameter of the graph.

Open Problem 4. *Construct a $(1+\varepsilon)$-approximate routing scheme for a weighted unit-disk graph that uses labels, routing tables, and headers of $O(\log n)$ bits.*

The SSSP problem in weighted unit-disk graphs needs heavy machinery, namely data structures for dynamic nearest-neighbor searching [Cha10b], and requires nearly $O\left(n \log^{12} n\right)$ time. However, for the unweighted case we have presented in Chapter 3 a simple approach that solves the problem in linear time, after presorting the disk centers. Moreover, in weighted planar graphs, the SSSP algorithm of Henzinger et al. [HKRS97] takes only linear time.

Open Problem 5. *Solve SSSP in a weighted unit-disk graph in $O(n \log n)$ time.*

So far we studied only planar graphs with non-negative edge weights, where the algorithm of Henzinger et al. [HKRS97] addresses SSSP in linear time. However, for the more general case of negative edge weights, the best SSSP algorithm, due to Klein et al.[KMW10], runs in $O\left(n \frac{\log^2 n}{\log \log n}\right)$ time.

Open Problem 6. *Solve SSSP in a planar graph with negative edge weights in linear time.*

It is interesting to investigate whether we can remove the two logarithmic factors from our $(1+\varepsilon)$-approximation algorithm of Chapter 5 for the diameter of a non-negatively-weighted, undirected planar graph. The first logarithmic factor stems from adapting the recursive scheme of Weimann and Yuster [WY16] of $O(\log n)$ depth, but there exist recursive approaches in planar graphs that require only a sublogarithmic number of levels. For example, the SSSP algorithm and the $(1+\varepsilon)$-approximate distance oracle of Henzinger et al. [HKRS97] and Kawarabayashi et al. [KST13] respectively use a recursion of only $O(\log n)$ levels. The second logarithmic factor of our algorithm originates from the multiple shortest paths data structure of Klein [Kle05], which we used to repeatedly compute shortest-path distances between a small number of pairs of vertices.

Open Problem 7. *Compute a $(1+\varepsilon)$-approximation of the diameter of a non-negatively-weighted, undirected planar graph in $o\left(n \log^2 n\right)$ time, where $\varepsilon > 0$ is a constant.*

Our diameter algorithm of Chapter 5 works only for the undirected case, so extending it for directed planar graphs would be a natural research direction. Our algorithm relies heavily on a Voronoi-diagram-based technique and on the framework of Weimann and Yuster's solution. While the former works in the directed setting as well, the latter does not. The first issue is that there is no known linear-time algorithm that computes an $\Theta(1)$-approximation of the diameter of a directed planar graph, but it is likely that such an algorithm exists. The second issue is that given a shortest path separator in a directed planar graph, it is not known how to approximate the length of the longest shortest path between two vertices in different sides of the separator in near linear time. To circumnavigate that issue, we could use techniques similar to those of Thorup [Tho04a] for constructing $(1+\varepsilon)$-approximate distance oracles for weighted, directed planar graphs.

Open Problem 8. *Compute a $(1+\varepsilon)$-approximation of the diameter of a non-negatively weighted, directed planar graph in near-linear time.*

In Chapter 5, we also presented a $(1+\varepsilon)$-approximate distance oracle of $O(\log(1/\varepsilon))$ query time and $\Theta((1/\varepsilon)^{O(1)}n)$ space. One could try to further reduce the query time to constant, which is what the oracle of Gu and Xu [GX15] achieves, but by increasing the dependency on $1/\varepsilon$ in its space to exponential. Another direction worth considering would be improving the space to linear, as the oracle of Kawarabayashi et al. [KKS11] does, but at the expense of polylogarithmic in n query time.

Open Problem 9. *Construct a $(1+\varepsilon)$-approximate distance oracle for a non-negatively weighted, undirected planar graph with $\Theta((1/\varepsilon)^{O(1)}n)$ space and $O(1/\varepsilon)$ query time or with linear space and $O(\log(1/\varepsilon))$ query time.*

In Chapter 6, we constructed a $(1+\varepsilon)$-approximate distance oracle for a weighted unit-disk graph with $O(1/\varepsilon)$ query time and near $O(n\log^2 n)$ space. One logarithmic factor stems from employing sparse neighborhood covers and another from storing $(1+\varepsilon)$-approximations of the distances of roughly $O(n\log n)$ pairs of vertices. These pairs are produced by constructing a WSPD with the algorithm of Gao and Zhang [GZ05].

Open Problem 10. *Construct a $(1+\varepsilon)$-approximate distance oracle for a weighted unit-disk graph with linear space and $O(1/\varepsilon)$ query time.*

Our algorithm of Chapter 4 for the APSP problem in unweighted intersection graphs of arbitrary line segments requires $O(n^{7/3}\log^{1/3} n)$ time. That is because we employed the data structure of Chazelle [Cha93] for intersection detection of line segments. However, it is likely that using instead techniques similar to those of Matoušek [Mat93] for Hopcroft's problem, the time could be improved to $O(n^{7/3}\log\log n)$ or better.

Open Problem 11. *Solve APSP in an unweighted intersection graph of arbitrary line segments in $o(n^{7/3}\log^{1/3} n)$ time.*

Finally, it is worth mentioning that the focus of this thesis was on theoretical results. As explained in Chapter 1, many real-world applications, such as geographical information systems and ad-hoc communication networks, utilize shortest-path algorithms and data structures. Thus, another research direction is to find more practical variants of our techniques.

References

[AAAS94] Pankaj K. Agarwal, Noga Alon, Boris Aronov, and Subhash Suri. Can visibility graphs be represented compactly? *Discrete & Computational Geometry*, 12(3):347–365, 1994.

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, 1998.

[ACC 96] Srinivasa Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proceedings of the 4th European Symposium on Algorithms (ESA)*, pages 514–528, 1996.

[ACD 16] Ittai Abraham, Shiri Chechik, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. On dynamic approximate shortest paths for planar graphs with worst-case costs. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 740–753, 2016.

[ACG12] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 1199–1218, 2012.

[ACIM99] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[ACT14] Peyman Afshani, Timothy M. Chan, and Konstantinos Tsakalidis. Deterministic rectangle enclosure and offline dominance reporting on the RAM. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 77–88, 2014.

[ADD 93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[AE99] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

[AF90] Miklos Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *The Journal of Symbolic Logic*, 55(1):113–150, 1990.

[AGM97] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *Journal of Computer and System Sciences*, 54(2):255–262, 1997.

[AGW15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1681–1697, 2015.

[AHLT05] Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005.

[AMOT90] Ravindra K. Ahuja, Kurt Mehlhorn, James Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM*, 37(2):213–223, 1990.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 503–513, 1990.

[APS93] Pankaj K. Agarwal, Marco Pellegrini, and Micha Sharir. Counting circular arc intersections. *SIAM Journal on Computing*, 22(4):778–793, 1993.

[BCPDB03] John M. Boyer, Pier Francesco Cortese, Maurizio Patrignani, and Giuseppe Di Battista. Stop minding your *P*'s and *Q*'s: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In *Proceedings of the 11th International Symposium on Graph Drawing (GD)*, pages 25–36, 2003.

[BDEG15] Michael J. Bannister, William E. Devanny, David Eppstein, and Michael T. Goodrich. The galois complexity of graph drawing: Why numerical solutions are ubiquitous for force-directed, spectral, and circle packing drawings. *Journal of Graph Algorithms and Applications*, 19(2):619–656, 2015.

[BFLO06] Krists Boitmanis, Kārlis Freivalds, Pēteris Lediņš, and Rūdolfs Opmanis. Fast and simple approximation of the diameter and radius of a graph. In *Proceedings of the 5th International Workshop on Experimental and E cient Algorithms (WEA)*, pages 98–108, 2006.

[BGKS15] Maxim A. Babenko, Pawel Gawrychowski, Tomasz Kociumaka, and Tatiana A. Starikovskaya. Wavelet trees meet suffix trees. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 572–591, 2015.

[BK95] Heinz Breu and David G. Kirkpatrick. On the complexity of recognizing intersection and touching graphs of disks. In *Proceedings of the 2nd International Symposium on Graph Drawing (GD)*, pages 88–98, 1995.

[BK98] Heinz Breu and David G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

[BK07] Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In *Proceedings of the 10th International Symposium on Algorithms and Data Structures (WADS)*, pages 541–552, 2007.

[BK10] Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[Ble08] Guy E. Blelloch. Space-efficient dynamic orthogonal point location, segment intersection, and range reporting. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 894–903, 2008.

[BLT07] Costas Busch, Ryan LaFortune, and Srikanta Tirthapura. Improved sparse covers for graphs excluding a fixed minor. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 61–70, 2007.

[BM04] John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(2):241–273, 2004.

[BM11] Kevin Buchin and Wolfgang Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *Journal of the ACM*, 58(2):6, 2011.

[BMN 04] Prosenjit Bose, Anil Maheshwari, Giri Narasimhan, Michiel H. M. Smid, and Norbert Zeh. Approximating geometric bottleneck shortest paths. *Computational Geometry*, 29(3):233–249, 2004.

[BS06] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Transactions on Algorithms (TALG)*, 2(4):557–577, 2006.

[Cab06] Sergio Cabello. Many distances in planar graphs. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 1213–1220, 2006.

[Cab12] Sergio Cabello. Many distances in planar graphs. *Algorithmica*, 62(1-2):361–381, 2012.

[Cab17a] Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2143–2152, 2017.

[Cab17b] Sergio Cabello. Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs. *CoRR*, abs/1702.07815v1, 2017.

[CDW17] Vincent Cohen-Addad, Søren Dahlgaard, and Christian Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. In *Proceedings of the 58th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 962–973, 2017.

[CE01] Timothy M. Chan and Alon Efrat. Fly cheaply: On the minimum fuel consumption problem. *Journal of Algorithms*, 41(2):330–337, 2001.

[CGS99] Boris V. Cherkassky, Andrew V. Goldberg, and Craig Silverstein. Buckets, heaps, lists, and monotone priority queues. *SIAM Journal on Computing*, 28(4):1326–1346, 1999.

[Cha93] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

[Cha08] Timothy M. Chan. All-pairs shortest paths with real weights in $O(n^3 \{ \log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[Cha10a] Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *Journal of the ACM*, 57(3):16:1–16:15, 2010.

[Cha10b] Timothy M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39(5):2075–2089, 2010.

[Cha12] Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8:1–17, 2012.

[Cha13] Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, 9(3):22, 2013.

[Che14] Shiri Chechik. Approximate distance oracles with constant query time. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 654–663, 2014.

[CJ15] Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015.

[CK95] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67–90, 1995.

[CL15] Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the 47th ACM Symposium on Theory of Computing (STOC)*, pages 31–40, 2015.

[CLP11] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

[CLR 14] Shiri Chechik, Daniel H. Larkin, Liam Roditty, Grant Schoenebeck, Robert E. Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1041–1052, 2014.

[CM11] Bernard Chazelle and Wolfgang Mulzer. Computing hereditary convex structures. *Discrete & Computational Geometry*, 45(4):796–823, 2011.

[CMSV17] Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $O\ m^{10\{7}\log W$ time. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 752–771, 2017.

[Coh98] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998.

[Coz82] Margaret B. Cozzens. *Higher and Multi-Dimensional Analogues of Interval Graphs*. PhD thesis, Rutgers The State University of New Jersey, 1982.

[CP10] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010.

[CR10] Sergio Cabello and Günter Rote. Obnoxious centers in graphs. *SIAM Journal on Discrete Mathematics*, 24(4):1713–1730, 2010.

[CS89] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4(5):387–421, 1989.

[CS03] Charles R. Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003.

[CS16] Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

[CS17a] Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in geometric intersection graphs. In *Proceeding of the 15th International Symposium on Algorithms and Data Structures (WADS)*, pages 253–264, 2017.

[CS17b] Timothy M. Chan and Dimitrios Skrepetos. Faster approximate diameter and distance oracles in planar graphs. In *Proceeding of the 25th European Symposium on Algorithms (ESA)*, pages 25:1–25:13, 2017.

[CS18] Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. In *Proceedings of the 34th Annual ACM Symposium on Computational Geometry (SoCG)*, 2018. (to appear).

[CT18] Timothy M. Chan and Konstantinos Tsakalidis. Dynamic planar orthogonal point location in sublogarithmic time. In *Proceedings of the 34th Annual ACM Symposium on Computational Geometry (SoCG)*, 2018. (to appear).

[CW16] Timothy M. Chan and Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

[CX00] Danny Z. Chen and Jinhui Xu. Shortest path queries in planar graphs. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 469–478, 2000.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

[dFdM12] Hubert de Fraysseix and Patrice Ossona de Mendez. Trémaux trees and planarity. *European Journal of Combinatorics*, 33(3):279–293, 2012.

[dFdMR06] Hubert de Fraysseix, Patrice Ossona de Mendez, and Pierre Rosenstiehl. Trémaux trees and planarity. *International Journal of Foundations of Computer Science*, 17(5):1017–1030, 2006.

[DFPP90] Hubert De Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[DHZ00] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[Dia69] Robert B. Dial. Algorithm 360: Shortest-path forest with topological ordering [H]. *Communications of the ACM*, 12(11):632–633, 1969.

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[Dji96] Hristo N. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 151–165, 1996.

[Dob90] Wlodzimierz Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *International Journal of Computer Mathematics*, 32(1-2):49–60, 1990.

[DP08] Ran Duan and Seth Pettie. Bounded-leg distance and reachability oracles. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 436–445, 2008.

[DR18] Ran Duan and Hanlin Ren. Approximating all-pair bounded-leg shortest path and apsp-af in truly-subcubic time. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, 2018. (to appear).

[DSST89] James R. Driscoll, Neil Sarnak, Daniel D. Sleator, and Robert E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[EGS86] Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

[EM81] Herbert Edelsbrunner and Hermann A. Maurer. On the intersection of orthogonal objects. *Information Processing Letters*, 13(4/5):177–181, 1981.

[EM01] David Eppstein and S. Muthukrishnan. Internet packet filter management and rectangle geometry. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 827–835, 2001.

[EM13] Esther Ezra and Wolfgang Mulzer. Convex hull of points lying on lines in $O(n \log n)$ time after preprocessing. *Computational Geometry*, 46(4):417–434, 2013.

[EMT95] David Eppstein, Gary L. Miller, and Shang-Hua Teng. A deterministic linear time algorithm for geometric separators and its applications. *Fundamenta Informaticae*, 22(4):309–329, 1995.

[EO82] Herbert Edelsbrunner and Mark H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14(3):124–127, 1982.

[Epp99] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of Graph Algorithms and Applications*, 3(3):283–309, 1999.

[FM95] Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51(2):261–272, 1995.

[For87] Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *Journal of Computer and System Sciences*, 72(5):868–889, 2006.

[Fre76] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM Journal on Computing*, 5:83–89, 1976.

[Fre87] Greg N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[FT87] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[FW93] Michael L. Fredman and Dan E. Willard. Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47(3):424–436, 1993.

[FW94] Michael L. Fredman and Dan E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994.

[Gar85] Harold N. Garbow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148–168, 1985.

[GJSD97] Prosenjit Gupta, Ravi Janardan, Michiel H. M. Smid, and Bhaskar DasGupta. The rectangle enclosure and point-dominance problems revisited. *International Journal of Computational Geometry and Applications*, 7(5):437–455, 1997.

[GK09] Yoav Giora and Haim Kaplan. Optimal dynamic vertical ray shooting in rectilinear planar subdivisions. *ACM Transactions on Algorithms*, 5(3):28, 2009.

[GKM 18] Pawel Gawrychowski, Haim Kaplan, Shay Mozes, Micha Sharir, and Oren Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic Ø $n^{5{3}}$ time. In *Proceedings of the 29th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 495–514, 2018.

[GM97] Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges. *Information and Computation*, 134(2):103–139, 1997.

[GMWWN18] Pawel Gawrychowski, Shay Mozes, Oren Weimann, and Christian Wulff-Nilsen. Better tradeoffs for exact distance oracles in planar graphs. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 515–529, 2018.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[Goo95] Michael T. Goodrich. Planar separators and parallel polygon triangulation. *Journal of Computer and System Sciences*, 51(3):374–389, 1995.

[Gra72] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[GT89] Harold N. Gabow and Robert E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[GX15] Qian-Ping Gu and Gengchun Xu. Constant query time p1 "q-approximate distance oracle for planar graphs. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC)*, pages 625–636, 2015.

[GZ05] Jie Gao and Li Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing*, 35(1):151–169, 2005.

[Hag00] Torben Hagerup. Improved shortest paths on the word RAM. In *Proceedings of the 27th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 61–72, 2000.

[Han01] Yijie Han. Improved fast integer sorting in linear space. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 793–796, 2001.

[Han04] Yijie Han. Improved algorithm for all pairs shortest paths. *Information Processing Letters*, 91(5):245–250, 2004.

[Han08] Yijie Han. An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest path. *Algorithmica*, 51(4):428–434, 2008.

[HK01] Petr Hliněnỳ and Jan Kratochvíl. Representing graphs by disks and balls (a survey of recognition-complexity results). *Discrete Mathematics*, 229(1-3):101–124, 2001.

[HKRS97] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

[HM85] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[HT74] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

[HT12] Yijie Han and Tadao Takaoka. An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *Proceedings of the 13th Scandinavian Symposium on Algorithm Theory (SWAT)*, pages 131–141, 2012.

[IK01] Alon Efrat, Alon Itai, and Matthew J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[INSWN11] Giuseppe F. Italiano, Yahav Nussbaum, Piotr Sankowski, and Christian Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC)*, pages 313–322, 2011.

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, 1977.

[Kat97] Matthew J. Katz. 3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry*, 8(6):299–316, 1997.

[KKS11] Ken-ichi Kawarabayashi, Philip N. Klein, and Christian Sommer. Linear-space approximate distance oracles for planar, bounded-genus and minor-free graphs. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 135–146, 2011.

[Kle89] Rolf Klein. *Concrete and Abstract Voronoi Diagrams*, volume 400 of *Lecture Notes in Computer Science*. Springer, 1989.

[Kle02] Philip N. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 820–827, 2002.

[Kle05] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 146–155, 2005.

[Kle08] Rolf Klein. Well separated pair decomposition. *Encyclopedia of Algorithms*, pages 1–5, 2008.

[KLN09] Rolf Klein, Elmar Langetepe, and Zahra Nilforoushan. Abstract Voronoi diagrams revisited. *Computational Geometry*, 42(9):885–902, 2009.

[KM89] Jan Kratochvíl and Jiří Matoušek. NP-hardness results for intersection graphs. *Commentationes Mathematicae Universitatis Carolinae*, 30(4):761–773, 1989.

[KM91] Jan Kratochvíl and Jiří Matoušek. String graphs requiring exponential representations. *Journal of Combinatorial Theory, Series B*, 53(1):1–4, 1991.

[KM94] Jan Kratochvíl and Jirí Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory, Series B*, 62(2):289–315, 1994.

[KM12] Ross J. Kang and Tobias Müller. Sphere and dot product representations of graphs. *Discrete & Computational Geometry*, 47(3):548–568, 2012.

[KMM93] Rolf Klein, Kurt Mehlhorn, and Stefan Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Computational Geometry*, 3(3):157–184, 1993.

[KMNS12] Haim Kaplan, Shay Mozes, Yahav Nussbaum, and Micha Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 338–355, 2012.

[KMR 17] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proceedings of the 28th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2495–2504, 2017.

[KMRS15] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and reachability oracles for directed transmission graphs. In *Proceedings of the 31st International Symposium on Computational Geometry (SoCG)*, volume 34, 2015.

[KMRS18] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Routing in unit disk graphs. *Algorithmica*, 80(3):830–848, 2018.

[KMS13] Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 505–514, 2013.

[KMW10] Philip N. Klein, Shay Mozes, and Oren Weimann. Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm. *ACM Transactions on Algorithms (TALG)*, 6(2):30, 2010.

[Kra94] Jan Kratochvíl. A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics*, 52(3):233–252, 1994.

[KS98] Philip N. Klein and Sairam Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

[KST13] Ken-ichi Kawarabayashi, Christian Sommer, and Mikkel Thorup. More compact oracles for approximate distances in undirected planar graphs. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 550–563, 2013.

[LCW02] Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *Proceedings of the 21st Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1268–1277, 2002.

[Le 14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

[LG12] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proceedings of the 53rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 514–523, 2012.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Mat93] Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(2):157–182, 1993.

[Mat96] Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93(1):333–344, 1996.

[Mil86] Gary L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32(3):265–279, 1986.

[MM13] Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B*, 103(1):114–143, 2013.

[Moh93] Bojan Mohar. A polynomial time circle packing algorithm. *Discrete Mathematics*, 117(1-3):257–263, 1993.

[MS12] Shay Mozes and Christian Sommer. Exact distance oracles for planar graphs. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–222, 2012.

[MTV91] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 538–547, 1991.

[Mul94] Ketan Mulmuley. *Computational geometry - an introduction through randomized algorithms*. Prentice Hall, 1994.

[MWN10] Shay Mozes and Christian Wulff-Nilsen. Shortest paths in planar graphs with real lengths in $O(n \log^2 n / \log\log n)$ time. In *Proceedings of the 18th European Symposium on Algorithms (ESA)*, pages 206–217, 2010.

[Nus11] Yahav Nussbaum. Improved distance queries in planar graphs. In *Proceedings of the 12th International Symposium on Algorithms and Data Structures (WADS)*, pages 642–653, 2011.

[Ola90] Stephan Olariu. A simple linear-time algorithm for computing the center of an interval graph. *International Journal of Computer Mathematics*, 34(3-4):121–128, 1990.

[OvL81] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23:166–204, 1981.

[Păt11] Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011.

[Pet04] Seth Pettie. A new approach to all-pairs shortest paths on real-weighted graphs. *Theoretical Computer Science*, 312(1):47–74, 2004.

[PR05] Seth Pettie and Vijaya Ramachandran. A shortest path algorithm for real-weighted undirected graphs. *SIAM Journal on Computing*, 34(6):1398–1431, 2005.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Texts and Monographs in Computer Science. Springer, 1985.

[Ram97] Rajeev Raman. Recent results on the single-source shortest paths problem. *ACM SIGACT News*, 28(2):81–87, 1997.

[RS11] Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 261–272, 2005.

[RVW13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 515–524, 2013.

[San05] Piotr Sankowski. Shortest paths in matrix multiplication time. In *Proceedings of the 13th European Symposium on Algorithms (ESA)*, pages 770–778, 2005.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 138–148, 1990.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[Som14] Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys*, 46(4):45, 2014.

[SSŠ03] Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. Recognizing string graphs in NP. *Journal of Computer and System Sciences*, 67(2):365–380, 2003.

[ST83] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[ST99] Alan P. Sprague and Tadao Takaoka. $O(1)$ query time algorithm for all pairs shortest distances on interval graphs. *International Journal of Foundations of Computer Science*, 10(04):465–472, 1999.

[Sub95] Sairam Subramanian. *Parallel and dynamic shortest-path algorithms for sparse graphs*. PhD thesis, Brown University, 1995.

[SZ99] Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 605–614, 1999.

[Tak92] Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Information Processing Letters*, 43(4):195–199, 1992.

[Tak98] Tadao Takaoka. Subcubic cost algorithms for the all pairs shortest path problem. *Algorithmica*, 20(3):309–318, 1998.

[Tak04] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *Proceedings of the 10th International Conference on Computing and Combinatorics (COCOON)*, pages 278–289, 2004.

[Tak05] Tadao Takaoka. An $O(n^3 \log\log n/\log n)$ time algorithm for the all-pairs shortest path problem. *Information Processing Letters*, 96(5):155–161, 2005.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[Tho00] Mikkel Thorup. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109, 2000.

[Tho03] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In *Proceedings of the 31st ACM Symposium on Theory of Computing (STOC)*, pages 149–158, 2003.

[Tho04a] Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *Journal of the ACM*, 51(6):993–1024, 2004.

[Tho04b] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. *Journal of Computer and System Sciences*, 69(3):330–353, 2004.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[Vas12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In *Proceedings of the 44th Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

[vEB77] Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information processing letters*, 6(3):80–82, 1977.

[vEBKZ76] Peter van Emde Boas, Robert Kaas, and Erik Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10(1):99–127, 1976.

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

[Wil14] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 664–673, 2014.

[WN10] Christian Wulff-Nilsen. *Algorithms for planar graphs and graphs in metric spaces*. PhD thesis, University of Copenhagen, 2010.

[WN13a] Christian Wulff-Nilsen. Approximate distance oracles with improved query time. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 539–549, 2013.

[WN13b] Christian Wulff-Nilsen. Constant time distance queries in planar unweighted graphs with subquadratic preprocessing time. *Computational Geometry*, 46(7):831–838, 2013.

[WY16] Oren Weimann and Raphael Yuster. Approximating the diameter of planar graphs in near linear time. *ACM Transactions on Algorithms (TALG)*, 12(1):12, 2016.

[Yan82] Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.

[YXD12] Chenyu Yan, Yang Xiang, and Feodor F. Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Computational Geometry*, 45(7):305–325, 2012.

[Zwi01] Uri Zwick. Exact and approximate distances in graphs – a survey. In *Proceedings of the 9th European Symposium on Algorithms (ESA)*, pages 33–48, 2001.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.

[Zwi06] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.