



Faster Algorithms for Reverse Shortest Path in Unit-Disk Graphs and Related Geometric Optimization Problems: Improving the Shrink-And-Bifurcate Technique

Timothy M. Chan  

Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, IL, USA

Zhengcheng Huang  

Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign, IL, USA

Abstract

In a series of papers, Avraham, Filtser, Kaplan, Katz, and Sharir (SoCG'14), Kaplan, Katz, Saban, and Sharir (ESA'23), and Katz, Saban, and Sharir (ESA'24) studied a class of geometric optimization problems – including reverse shortest path in unweighted and weighted unit-disk graphs, discrete Fréchet distance with one-sided shortcuts, and reverse shortest path in visibility graphs on 1.5-dimensional terrains – for which standard parametric search does not work well due to a lack of efficient parallel algorithms for the corresponding decision problems. The best currently known algorithms for all the above problems run in $O^*(n^{6/5}) = O^*(n^{1.2})$ time (ignoring subpolynomial factors), and they were obtained using a technique called *shrink-and-bifurcate*. We improve the running time to $\tilde{O}(n^{8/7}) \approx O(n^{1.143})$ for these problems. Furthermore, specifically for reverse shortest path in unweighted unit-disk graphs, we improve the running time further to $\tilde{O}(n^{9/8}) = \tilde{O}(n^{1.125})$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Geometric optimization problems, parametric search, shortest path, disk graphs, Fréchet distance, visibility, distance selection, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.32

Related Version *Full Version*: <https://arxiv.org/abs/2504.06434>

Funding *Timothy M. Chan*: Work supported by NSF Grant CCF-2224271.

1 Introduction

Parametric search is one of the most well-known and powerful techniques for solving geometric optimization problems [5]. First introduced by Megiddo [26] in the 80s, the technique has since found countless applications in computational geometry and beyond. To search for the optimal value r^* of an optimization problem, we first solve the decision problem: given an input value r , decide whether $r^* \leq r$. The idea is to simulate an algorithm \mathcal{A} for the decision problem¹ with the input value being the unknown r^* itself; whenever \mathcal{A} makes a comparison with the unknown r^* , we resolve the comparison by calling the decision algorithm. Megiddo showed that if \mathcal{A} is parallelizable with polylogarithmic number of rounds, then the simulation can be done efficiently with only a polylogarithmic factor increase in the total running time.

Over the years, a number of refinements and alternatives to parametric search have been proposed (e.g., Cole's trick [14], matrix searching [18, 19], expander-based approaches [24], Chan's randomized optimization technique [8] and implicit linear programming technique [10, 11]), but these alternatives are more specialized and are mainly about lowering the extra polylogarithmic factors in the running time.

¹ The algorithm \mathcal{A} that we are simulating need not solve the decision problem, although it often is the case, so long as the optimal value r^* is one of its “critical values”.



© Timothy M. Chan and Zhengcheng Huang;
licensed under Creative Commons License CC-BY 4.0

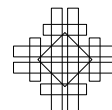
41st International Symposium on Computational Geometry (SoCG 2025).

Editors: Oswin Aichholzer and Haitao Wang; Article No. 32; pp. 32:1–32:14

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



In the last couple of years, Avraham, Filtser, Kaplan, Katz, and Sharir [6], Wang and Zhao [29], Kaplan, Katz, Saban, and Sharir [21], Katz, Saban, and Sharir [23], and Agarwal, Kaplan, Katz, and Sharir [3] have identified and explored a natural class of geometric optimization problems (listed below) where parametric search does not do too well. For these problems, efficient near-linear-time decision algorithms exist, but they do not seem efficiently parallelizable (the decision version of these problems are related to single-source shortest paths, but breadth-first search (BFS) and Dijkstra’s algorithm are inherently sequential). For many of the problems in this class, the search spaces are actually Euclidean distances defined by pairs of input points, and so parametric search could be replaced by binary search using an oracle for *distance selection*, but unfortunately, known distance selection algorithms requires near $O(n^{4/3})$ time in the plane [1, 24, 28, 13].

To overcome these issues, Avraham *et al.* [6] introduced a new clever variant of the parametric search technique they termed *shrink-and-bifurcate*, which was subsequently adopted by Kaplan *et al.* [21], Katz *et al.* [23], and Agarwal *et al.* [3]. The technique is able to solve most of these problems with an unusual running time between near-linear and $n^{4/3}$: namely, $O^*(n^{6/5})$.²

Although Avraham *et al.*’s improvement may seem incremental, this $O^*(n^{6/5})$ bound has remained stagnant (Avraham *et al.*’s paper appeared over 10 years ago), despite more applications and more appearances of this bound in recent papers [21, 23, 3]; the ideal goal of obtaining near-linear time at the moment seems far out of reach. In this work, we want to understand this class of problems better, and to see if this mysterious exponent $6/5$ can be improved.

In the following, we state the specific problems considered in this class and review their background.

Reverse shortest path in (unweighted) unit-disk graphs. We start with the simplest of these problems: *reverse shortest path* for unit-disk graphs. For every set of points P in the plane and positive real number r , let $G_r(P)$ denote the graph with vertex set P , where we put an edge between two points iff their Euclidean distance is at most r ; equivalently, $G_r(P)$ is the intersection graph of the disks with centers P and radius $r/2$. Note that $G_r(P)$ is a *unit-disk graph* (since we can rescale the radius to 1). In the reverse shortest path problem, one is given a set of points P in the plane, two vertices $s, t \in P$, and a positive integer λ , and is asked to compute the minimum real number r^* such that $G_{r^*}(P)$ contains an s -to- t path with at most λ edges.

This problem is quite natural and can be viewed equivalently as finding the minimum bottleneck path from s to t that uses at most λ links (or “hops”), where minimizing *bottleneck* refers to minimizing the longest edge in the path. (As is well known, computing minimum bottleneck paths without constraints on the number of links reduces to minimum spanning trees; *e.g.*, see [20]. It is reasonable to add restrictions on the number of links.)

The decision version of this problem takes an extra parameter r , and asks one to determine whether $G_r(P)$ contains an s -to- t -path with at most λ edges. The decision problem (or computing the minimum-link s -to- t path in a unit disk graph) can be solved in $O(n \log n)$ time, as shown by Cabello and Jejčić [7] using the Delaunay triangulation, or alternatively by Chan and Skrepetos [12] using a grid-based approach, or even earlier by Efrat, Itai, and Katz [17] (who solved a very similar problem as a subroutine for computing bichromatic maximum matchings). Now, since r^* is always the Euclidean distance between two input

² In this paper, we use the O^* and \tilde{O} notation to hide $n^{o(1)}$ and $(\log n)^{O(1)}$ factors respectively.

points, it was immediately noted by Cabello and Jeřič that reverse shortest path can be solved in $\tilde{O}(n^{4/3})$ time by binary search, using an $\tilde{O}(n^{4/3})$ -time distance selection algorithm (as we have mentioned earlier) to guide the search.

The first algorithm to beat $n^{4/3}$ for reverse shortest path was discovered by Wang and Zhao [29], using a grid-based approach. Their algorithm runs in $\tilde{O}(n^{5/4})$ time. Subsequently, Kaplan *et al.* [21] presented a different, faster randomized algorithm running in $O^*(n^{6/5})$ time, by applying the aforementioned shrink-and-bifurcate technique.

Reverse shortest path in weighted unit-disk graphs. In a *weighted* unit-disk graph, each edge is weighted by the Euclidean distance between the centers of the two disks. The reverse shortest path problem can be defined in a similar way, except that λ is now a cap on the sum of the weights along the s -to- t path.

Wang and Xue [27] gave an $O(n \log^2 n)$ -time algorithm for the decision problem in the weighted case. Wang and Zhao [29] extended their $\tilde{O}(n^{5/4})$ -time algorithm for reverse shortest path to weighted unit-disk graphs (with a slightly larger polylogarithmic factor). Kaplan *et al.* [21] similarly extended their $O^*(n^{6/5})$ -time randomized algorithm to the weighted case, again via shrink-and-bifurcate.

Reverse shortest path in unweighted and weighted disk graphs. The reverse shortest path problem can be generalized to intersection graphs of disks with arbitrary radii. The input is a set S of disks in the plane, and $G_r(S)$ is defined as the graph with S as vertices, where we put an edge between two disks iff their distance is at most r (this is sometimes referred to as a *proximity graph*); equivalently, $G_r(S)$ is the intersection graph of the disks with the same centers as S but radii increased additively by $r/2$. In the weighted case, each edge is weighted by the Euclidean distance between the centers of the two disks (or alternatively, the Euclidean distance of the centers minus the sum of the two radii).

As shown by Kaplan *et al.* [21], the decision problem can be solved in $\tilde{O}(n)$ time by a careful implementation of BFS (in the unweighted case) or Dijkstra's algorithm (in the weighted case), using known dynamic geometric data structures [22] (which have large hidden polylogarithmic factors).

Wang and Zhao's grid-based approach to reverse shortest path does not extend to arbitrary disk graphs. However, the shrink-and-bifurcate approach is still applicable, and yields an $O^*(n^{5/4})$ -time randomized algorithm as shown by Kaplan *et al.* [21].

Reverse shortest path in segment proximity graphs. Another generalization is to the case of line segments. Given a set S of *disjoint* line segments in the plane, define the segment proximity graph $G_r(S)$ to be the graph with S as vertices, where we put an edge between two segments iff their distance is at most r . The reverse shortest path problem for (unweighted) line segments can then be defined in the same way.

This problem was first considered by Agarwal *et al.* [4], who gave a randomized $O^*(n^{4/3})$ -time algorithm. Later, Agarwal *et al.* [3] solved the corresponding decision problem in $O(n \log^2 n)$ -time. They then applied the shrink-and-bifurcate technique to obtain an improved randomized algorithm for the optimization problem running in $O^*(n^{6/5})$ time, assuming that the spread (the maximum-to-minimum distance ratio) is polynomially bounded.

Discrete Fréchet distance with one-sided shortcuts. The *discrete Fréchet distance* is a popular measure of closeness between two polygonal chains. Given polygonal chains $P = \langle p_1, \dots, p_{|P|} \rangle$ and $Q = \langle q_1, \dots, q_{|Q|} \rangle$ with $|P| + |Q| = n$, consider two frogs, a P -frog

and a Q -frog, starting at points p_1 and q_1 respectively. In each move, one frog leap from its current vertex to the next vertex on their respective polygonal chain, while the other frog stays at its current vertex. The discrete Fréchet distance between P and Q is defined as the minimum distance r^* for which the two frogs can arrive at $p_{|P|}$ and $q_{|Q|}$, such that the Euclidean distance between the two frogs is always at most r^* before and after every leap.

The discrete Fréchet distance with *one-sided shortcuts* is defined similarly, except that we allow one frog, say the P -frog, to leap to any vertex ahead during each move. In other words, the one-sided discrete Fréchet distance between P and Q is the minimum discrete Fréchet distance between P' and Q , over all subsequences P' of P that start at p_1 and end at $p_{|P|}$. The problem of computing the discrete Fréchet distance with one-sided shortcuts is what first prompted Avraham *et al.* [6] to introduce their shrink-and-bifurcate technique. They noted a linear-time algorithm for the decision problem and obtained an $O^*(n^{6/5})$ -time randomized algorithm for the optimization problem.

Reverse shortest path in visibility graphs over 1.5-dimensional terrains. A *terrain* T in 1.5 dimensions refers to an x -monotone polygonal chain in the plane. Two points p, q on or above the terrain are said to *see* each other if the line segment \overline{pq} does not intersect T . Given a 1.5-dimensional terrain T and a set of points P on or above T with $|P| + |T| = n$, the visibility graph $G(P, T)$ is a graph with vertices P and edges indicating which pairs of points see each other.

In the reverse shortest path problem over a terrain T , we are given a set P of points on T , and we want to erect a tower at each point in P . In particular, given a terrain T , a set of points P on T , a pair of points $s, t \in P$, and a positive integer λ , we are asked to find the minimum real number h^* such that the visibility graph $G(P(h^*), T)$ contains an s - t path with at most λ edges, where $P(h^*)$ is the tips of the towers of height h^* erected at the points of P .

This problem was first introduced by Agarwal *et al.* [4], who gave a randomized $O^*(n^{4/3})$ -time algorithm. Katz *et al.* [23] solved the decision problem in $\tilde{O}(n)$ time by a careful implementation of BFS on the visibility graph, with clever uses of range searching data structures. Katz *et al.* then applied the shrink-and-bifurcate technique to obtain a randomized $O^*(n^{6/5})$ -time algorithm for the optimization problem.

1.1 The Shrink-and-Bifurcate Technique

As noted, the current fastest algorithms for all of the above problems are obtained via the shrink-and-bifurcate technique introduced by Avraham *et al.* [6]. The setting requires that critical values (values that the decision algorithm may compare the input value against) are distances defined by pairs of input objects for some distance function. The strategy consists of two stages, *interval shrinking* and *bifurcation*.

1. The interval shrinking stage is a relaxed version of binary search – instead of insisting on finding the answer r^* immediately, we seek an interval $(r^-, r^+]$ that contains r^* and that contains at most L critical values for a given parameter L . In the case of Euclidean distances of points in the plane, Avraham *et al.* showed that this stage can be done in $O^*(n^{4/3}/L^{1/3} + D(n))$ time (using randomization), where $D(n)$ is the running time of the decision algorithm. This is potentially better than the $\tilde{O}(n^{4/3})$ time bound for distance selection.

2. The bifurcation stage then uses this interval $(r^-, r^+]$ to simulate the decision algorithm at the unknown r^* more efficiently than standard parametric search when the decision algorithm is not parallelizable. Avraham et al. showed that this simulation can be done in $\tilde{O}(L^{1/2}D(n))$ time.

When $D(n) = \tilde{O}(n)$, choosing $L = n^{2/5}$ to balance the costs of the two stages gives an overall running time of $O^*(n^{6/5})$.

Some of the applications use distance functions other than Euclidean distances of points in the plane, but the $O^*(n^{4/3}/L^{1/3} + D(n))$ bound for interval shrinking is still applicable. An exception is the reverse shortest path problem for general disk graphs. Here, the corresponding distance selection problem requires $O^*(n^{3/2})$ time, and the interval shrinking subproblem is solved in $O^*(n^{3/2}/L^{1/2} + D(n))$ time instead. When $D(n) = \tilde{O}(n)$, choosing $L = n^{1/2}$ to balance the costs of the two stages gives an overall running time of $O^*(n^{5/4})$ instead.

1.2 New Results

The new contributions in this paper are twofold:

1. We improve all the previous $O^*(n^{6/5})$ time bounds – for reverse shortest path in unit-disk graphs, weighted unit-disk graphs, segment proximity graphs, visibility graphs over 1.5-dimensional terrains, and discrete Fréchet distance with one-sided shortcuts – to $\tilde{O}(n^{8/7})$. (In particular, for discrete Fréchet distance with one-sided shortcuts, this is the first improvement on the exponent in 10 years.)

For reverse shortest path in arbitrary disk graphs (unweighted or weighted), we improve the previous $O^*(n^{5/4})$ time bound to $O^*(n^{6/5})$.

Although the improvements are incremental, they are all obtained from one simple idea for improving interval shrinking (we keep the bifurcation part unchanged) – the simplicity and generality of our idea make it easy to apply to existing (and potentially future) applications of the shrink-and-bifurcate approach, and are the main takeaways of the paper. More precisely, we show how to lower the $O^*(n^{4/3}/L^{1/3} + D(n))$ time bound for interval shrinking subproblem to $\tilde{O}(n^{4/3}/L^{2/3} + D(n))$ (or in the arbitrary disk graph case, lower $O^*(n^{3/2}/L^{1/2} + D(n))$ to $O^*(n^{3/2}/L^{3/4} + D(n))$). The original interval shrinking method by Avraham *et al.* needs to modify range searching techniques for distance selection, constructing “partial biclique covers” and then applying random sampling. Our new method, in contrast, can be described in just a few lines, and simply uses a known distance selection algorithm *as a black box* on random subsets (see Section 2).

2. Next, we focus more closely on the simplest of this class of problems, namely, reverse shortest path in (unweighted) unit-disk graphs. For this specific problem, we are able to further improve the running time to $\tilde{O}(n^{9/8})$. The improvement of the exponent from $6/5 = 1.2$ to $9/8 = 1.125$ is a bigger increment (for example, comparable in magnitude to Wang and Zhao’s original improvement [29] from $4/3 \approx 1.333$ to $5/4 = 1.25$, or Kaplan *et al.*’s improvement [21] from $5/4 = 1.25$ to $6/5 = 1.2$).

Interestingly, we obtain our result by combining the shrink-and-bifurcate technique with Wang and Zhao’s original grid-based approach! Although it may be natural to consider combining the two approaches, how to do so is not obvious at all. We need to put together several nontrivial technical ideas – this is the most intricate part of the paper (see Section 3).

2 Interval Shrinking

In this section, we formally define the *interval shrinking* subproblem in an abstract, self-contained setting, and then describe our new randomized method for this subproblem.

► **Subproblem 1 (Interval shrinking).** *Suppose we have an unknown value r^* . Suppose there is a decision oracle that can decide whether $r^* \leq r$ for an input value r in $D(n)$ time where $D(n) \geq n$.*

Let $\delta(\cdot, \cdot)$ be a function over pairs of objects. Suppose there is a selection oracle that, given two sets P and Q of objects and a number k , can compute the k -th smallest value in $\delta(P, Q) := \{\delta(p, q) \mid (p, q) \in P \times Q\}$ in $K(|P|, |Q|) = \tilde{O}(|P|^\alpha |Q|^\alpha + |P| + |Q|)$ time for some constant $\alpha < 1$.

Given two sets P and Q of objects with $|P| + |Q| = n$ and a number L , we want to compute an interval $(r^-, r^+]$ that contains r^ and that contains at most $\tilde{O}(L)$ values in $\delta(P, Q)$.*

If δ is the Euclidean distance function for points in the plane (which is the case for the applications to reverse shortest path in unit-disk graphs or discrete Fréchet distance with one-sided shortcuts), known distance selection algorithms [1, 24, 28, 13] between an m -point set and an n -point set run in $K(m, n) = \tilde{O}(m^{2/3}n^{2/3} + m + n)$ time, and thus we have $\alpha = 2/3$. In this planar Euclidean case, Avraham *et al.* [6] solved the above interval shrinking problem in $O^*(n^{4/3}/L^{1/3} + D(n))$ time with high probability. We will describe an improved method.

It suffices to describe how to find an $r^+ > r^*$ such that $[r^*, r^+]$ contains at most $\tilde{O}(L)$ values in $\delta(P, Q)$, since finding r^- is analogous. Also, it suffices to design an algorithm which is correct with probability $\Omega(1)$, since we can get an algorithm which is correct w.h.p.³ by repeating for logarithmically many trials and returning the minimum r^+ found.

2.1 First Version

Our first interval shrinking algorithm is simple, as presented in Algorithm 1.

■ **Algorithm 1** Basic interval shrinking.

-
- 1 Take random subsets $\hat{P} \subseteq P$ and $\hat{Q} \subseteq Q$ where each element is chosen independently with probability $\frac{1}{\sqrt{L}}$
 - 2 **return** the successor r^+ of r^* in $\delta(\hat{P}, Q) \cup \delta(P, \hat{Q})$, which can be computed by binary searches in $\delta(\hat{P}, Q)$ and $\delta(P, \hat{Q})$ using the selection and decision oracles
-

Correctness. Let $E \subseteq P \times Q$ be the set of L pairs that have the L smallest $\delta(\cdot, \cdot)$ values after r^* . Think of E as a bipartite graph between P and Q . Then E has at least \sqrt{L} non-isolated vertices in P or at least \sqrt{L} non-isolated vertices in Q (because otherwise, E would have fewer than L edges). W.l.o.g., assume the former. Then $\hat{P} \times Q$ hits E with probability at least $1 - (1 - \frac{1}{\sqrt{L}})^{\sqrt{L}} = \Omega(1)$. This implies that there are at most L values in $\delta(\hat{P}, Q)$ between r^* and r^+ with probability $\Omega(1)$.

³ With high probability, i.e., probability $1 - O(1/n^c)$ for an arbitrarily large constant c .

Running time. Note that $|\widehat{P}|, |\widehat{Q}| = \tilde{O}(\frac{n}{\sqrt{L}})$ w.h.p. Using the selection and decision oracles, w.h.p., the binary searches can be done in time

$$\tilde{O}\left(\left(\frac{n}{\sqrt{L}}\right)^\alpha n^\alpha + n^\alpha \left(\frac{n}{\sqrt{L}}\right)^\alpha + D(n)\right) = \tilde{O}(n^{2\alpha}/L^{\alpha/2} + D(n)).$$

For planar Euclidean distances with $\alpha = 2/3$, we have thus recovered the $O^*(n^{4/3}/L^{1/3} + D(n))$ result of Avraham *et al.* [6]. However, our algorithm is arguably simpler, because, unlike Avraham *et al.*, we invoke distance selection as a black box, without needing to modify the internal mechanisms of distance selection.

We remark that the idea of randomly sampling *pairs* of input objects is more common and has been used before in the context of slope selection or distance selection [25, 16, 9] (and in Avraham *et al.*'s previous method [6]). But here we are sampling the input objects themselves (which in some sense is even simpler, though the analysis is slightly trickier).

2.2 Improved Version

Upon closer examination, one may notice that the basic algorithm has room for improvement. In particular, if E has $\Theta(\sqrt{L})$ non-isolated vertices in P and $\Theta(\sqrt{L})$ non-isolated vertices in Q , then clearly our samples $\widehat{P} \times Q$ and $P \times \widehat{Q}$ are wasteful. Indeed, we can eliminate such redundancy by sampling in P and Q simultaneously with different rates. We present the details in Algorithm 2.

■ **Algorithm 2** Improved interval shrinking.

```

1 for  $i = 1, \dots, \log n$  do
2   Take a random subset  $\widehat{P}_i \subseteq P$  where each element is chosen independently with
   probability  $\frac{1}{\lceil L/2^i \rceil}$ 
3   Take a random subset  $\widehat{Q}_i \subseteq Q$  where each element is chosen independently with
   probability  $\frac{1}{2^i}$ 
4 end
5 return the successor  $r^+$  of  $r^*$  in  $\bigcup_{i=1}^{\log n} \delta(\widehat{P}_i, \widehat{Q}_i)$ , which can be computed by binary
   search in each  $\delta(\widehat{P}_i, \widehat{Q}_i)$  using the selection and decision oracles

```

Correctness. Let $E \subseteq P \times Q$ be the set of $L \log n$ pairs that have the $L \log n$ smallest $\delta(\cdot, \cdot)$ values after r^* . Consider the bipartite graph $H = (P \sqcup Q, E)$. Let $M_i \subseteq P$ be the vertices with degree in the range $[2^{i-1}, 2^i)$ in H . Since $\sum_{i=1}^{\log n} 2^i |M_i| \geq L \log n$, there must exist an index i_0 for which $|M_{i_0}| \geq \lceil L/2^{i_0} \rceil$. Observe that $\widehat{P}_{i_0} \times \widehat{Q}_{i_0}$ hits E if

- (i) \widehat{P}_{i_0} hits M_{i_0} , and
- (ii) for some $p \in \widehat{P}_{i_0} \cap M_{i_0}$, \widehat{Q}_{i_0} hits the neighborhood of p in E .

Now, (i) holds with probability at least $1 - (1 - \frac{1}{\lceil L/2^{i_0} \rceil})^{\lceil L/2^{i_0} \rceil} = \Omega(1)$. For a fixed $p \in \widehat{P}_{i_0} \cap M_{i_0}$, the probability that \widehat{Q}_{i_0} contains at least one neighbor of p is at least $1 - (1 - \frac{1}{2^{i_0}})^{2^{i_0-1}} = \Omega(1)$. Thus, conditioned on (i), the probability of (ii) is $\Omega(1)$. Hence, $\widehat{P}_{i_0} \times \widehat{Q}_{i_0}$ hits E with probability $\Omega(1)$. This implies that there are at most $L \log n$ values in $\delta(P, Q)$ between r^* and r^+ with probability $\Omega(1)$.

Running time. Note that $|\widehat{P}_i| = \tilde{O}(\frac{n}{\lceil L/2^i \rceil})$ and $|\widehat{Q}_i| = \tilde{O}(\frac{n}{2^i})$ w.h.p. Using the selection and decision oracles, w.h.p. the binary searches can be done in time

$$\tilde{O}\left(\sum_{i=1}^{\log n} \left(\left(\frac{n}{\lceil L/2^i \rceil}\right)^\alpha \left(\frac{n}{2^i}\right)^\alpha + D(n)\right)\right) = \tilde{O}(n^{2\alpha}/L^\alpha + D(n)).$$

► **Theorem 2.** *Subproblem 1 (interval shrinking) can be solved in $\tilde{O}(n^{2\alpha}/L^\alpha + D(n))$ time w.h.p.*

For example, for planar Euclidean distances, the bound is $\tilde{O}(n^{4/3}/L^{2/3} + D(n))$, which is noticeably better than $\tilde{O}(n^{4/3}/L^{1/3} + D(n))$.

2.3 Combining with Bifurcation

After interval shrinking, Avraham *et al.* [6] devised a clever variant of parametric search via an idea called *bifurcation* (also redescribed in more generality in Kaplan *et al.*'s subsequent paper [21]) which accomplishes the following:

► **Lemma 3** (Bifurcation [6]). *Suppose we have an unknown value r^* . Suppose there is a decision oracle that can decide whether $r^* \leq r$ for an input value r in $D(n)$ time.*

Suppose we have an algorithm \mathcal{A} that has one input parameter and runs in no more than $O(D(n))$ time, and during the execution of the algorithm, it can only access its input parameter via binary comparisons. The values that \mathcal{A} can compare its input parameter with are called the critical values.

Lastly, suppose that we are also given an interval $(r^-, r^+]$ that is guaranteed to contain r^ and that contains at most $\tilde{O}(L)$ critical values.*

Then we can simulate \mathcal{A} on input r^ in $\tilde{O}(L^{1/2}D(n))$ time.*

Typically, in applications of the technique, the algorithm \mathcal{A} we want to simulate is the decision algorithm itself; here, the simulation produces an interval containing r^* such that all input values lying in the interval have the same computation path and thus the same answer as r^* (namely, “yes”), and so the left endpoint of this interval gives us precisely the value of r^* .

We will not redescribe the proof of the above lemma, as we will only use it as a black box. (The rough idea in bifurcation is to simulate multiple branches of \mathcal{A} at the same time, until we have accumulated sufficiently many comparisons and will then resolve all of them as a batch using the decision oracle; comparisons with values outside the interval $(r^-, r^+]$ are free.) Combining our improved interval shrinking method with bifurcation, we immediately obtain better results for all optimization problems stated in the introduction:

► **Theorem 4.**

(a) *The reverse shortest path problem in unweighted and weighted unit-disk graphs, segment proximity graphs (assuming polynomial spread), and visibility graphs over 1.5-dimensional terrains, and the discrete Fréchet distance problem with one-sided shortcuts, as defined in the introduction, can all be solved in $\tilde{O}(n^{8/7})$ time w.h.p. by (Las Vegas) randomized algorithms.*

(b) *The reverse shortest path problem in unweighted and weighted disk graphs can be solved in $O^*(n^{6/5})$ time w.h.p. by (Las Vegas) randomized algorithms.*

Proof. As mentioned in the introduction, all these problems have known decision algorithms running in $D(n) = \tilde{O}(n)$ time [6, 21, 23, 3]. In all these decision algorithms, critical values are indeed obtained from some function δ over pairs of input objects.

For all the applications in (a), the function δ satisfies $\alpha = 2/3$. (Some care is needed for reverse shortest path in visibility graphs in 1.5-dimensional terrains; see the full version.)

Combining Theorem 2 with Lemma 3, we get total running time

$$\tilde{O}\left(\frac{n^{4/3}}{L^{1/3}} + L^{1/2}n\right).$$

Choosing $L = n^{2/7}$ yields $\tilde{O}(n^{8/7})$.

For the applications in (b), we have $\alpha = 3/4 + o(1)$ [21]. The total running time becomes

$$O^*\left(\frac{n^{3/2}}{L^{3/4}} + L^{1/2}n\right).$$

Choosing $L = n^{2/5}$ yields $O^*(n^{6/5})$. ◀

3 Reverse Shortest Path in Unweighted Unit-Disk Graphs

For reverse shortest path in unweighted unit-disk graphs, we show how to improve the running time even further.

In this section, $\delta(p, q)$ denotes the Euclidean distance between points p and q , and $\delta(P, Q) := \{\delta(p, q) \mid (p, q) \in P \times Q\}$ for two point sets P and Q . Let $d_G(s, t)$ denote the shortest-path distance between s and t in a graph G .

We are given an input set P of n points in the plane, $s, t \in P$, and a number λ . Let $G_r(P)$ be the (unweighted) graph with vertex set P , where we put an edge between p and q iff $\delta(p, q) \leq r$. We want to compute the smallest r^* for which $d_{r^*}(s, t) \leq \lambda$, where we use $d_r(s, t)$ as a shorthand for $d_{G_r(P)}(s, t)$.

3.1 Recap of Chan and Skrepetos' Decision Algorithm

Our algorithms rely on the details of the known $\tilde{O}(n)$ -time decision algorithm by Chan and Skrepetos [12], so we will first give a quick recap of their algorithm.

Given an input value r , we want to compute $d_r(s, t)$. We first create a uniform grid Ψ_r of side length $r/\sqrt{2}$. Chan and Skrepetos' algorithm is a careful implementation of BFS using the grid. We start from s and generate each level of the BFS tree (the "frontier") one at a time. The key observation is that each grid cell only needs to be visited a constant number of times during the BFS, because (i) points in the same cell have the roughly same shortest-path distance ± 1 from s in $G_r(P)$ and so each cell "appears" in only $O(1)$ levels of the BFS tree, and (ii) each cell has only $O(1)$ neighboring cells – we say that two cells are *neighboring* if the minimum Euclidean distance between the two cells is at most r . Given one level of the BFS tree, we can generate the next level (i.e., "expand" the frontier) by solving the following subproblems between cells in the current level and their neighboring cells:

► **Subproblem 5.** *Given an input value r , a subset of n_r red points in one grid cell σ and a subset of n_b blue points in a neighboring grid cell σ' , determine for each blue point whether there is a red point at distance at most r from it.*

Chan and Skrepetos solved the above subproblem in $O(n_r + n_b)$ time, assuming that the input points are pre-sorted, by using envelopes of pseudo-lines. As a result, the total running time of their algorithm is linear after pre-sorting.

The above algorithm is not efficiently parallelizable (since the number of levels in the BFS may be large). Instead, we can apply the bifurcation technique to simulate the algorithm on r^* . However, one issue arises: the envelope computations requires critical values defined

by triples of input elements, not pairs. As noted in other papers [21, 29], this can be fixed by switching to a different solution to Subproblem 5 which is slightly slower by a logarithmic factor but is simpler: namely, we compute the nearest red neighbor to each blue point by point location in the Voronoi diagram of the red points in $O((n_r + n_b) \log(n_r + n_b))$ time [15]; then for each blue point, we just compare r with its nearest neighbor distance. This way, the critical values are all Euclidean distances of pairs of input points.

Another issue is the construction of the grid Ψ_r , i.e., the assignment of input points to grid cells in Ψ_r . This part is easily parallelizable, and so we can apply standard parametric search to construct the grid Ψ_{r^*} in $\tilde{O}(n)$ time, all done as preprocessing. (In fact, the time bound for constructing Ψ_{r^*} is $O(n \log n)$, as shown by Wang and Zhao [29].)

3.2 Heavy vs. Light Cells

Applying the shrink-and-bifurcate technique to the preceding decision algorithm using our improved interval shrinking method yields an overall time bound of $\tilde{O}(n^{8/7})$, as we have already shown in Theorem 4(a). To obtain our best result, we will combine with another idea that was used in Wang and Zhao’s original paper [29]: heavy vs. light cells.

Recall that we have already constructed the grid Ψ_{r^*} . Let Δ be a parameter to be chosen later. Call a grid cell σ *heavy* if $|P \cap \sigma| \geq \Delta$, or *light* otherwise. The number of heavy cells is clearly at most $O(n/\Delta)$. Furthermore, a pair of neighboring cells (σ, σ') is called a *light pair* if one of σ and σ' is a light cell (the other may be light or heavy).

Our new strategy is as follows. First, we perform interval shrinking, but only with respect to distances of points from light pairs. Intuitively, there are fewer such distances and so interval shrinking should be doable faster. This allows us to simulate Chan and Skrepetos’ algorithm at r^* on a modified graph where heavy cells are eliminated – or more precisely, points in a common heavy cell are contracted into a single vertex. When we “uncontract” the heavy cells, this causes an $O(n/\Delta)$ additive error to all the shortest-path distances in $G_{r^*}(P)$. As a final step, we show how to use a dynamic program to recover the value r^* . In the next three subsections, we provide the details of all these steps.

3.3 Faster Interval Shrinking for Light Pairs

In the following, we show how to solve the interval shrinking problem faster for distances from light pairs. More precisely, define $\delta_{\text{light}}(p, q) = \delta(p, q)$ if $p \in \sigma$ and $q \in \sigma'$ for some light pair (σ, σ') , and $\delta_{\text{light}}(p, q) = \infty$ otherwise. We apply Algorithm 2 to do interval shrinking with respect to δ_{light} .

To analyze the running time, we need an efficient implementation of the selection oracle for

$$\delta_{\text{light}}(\hat{P}_i, \hat{Q}_i) = \bigcup_{\text{light pair } (\sigma, \sigma')} \delta(\hat{P}_i \cap \sigma, \hat{Q}_i \cap \sigma') \cup \{\infty\}.$$

Let $n_\sigma := |P \cap \sigma|$ for each cell σ . Note that $|\hat{P}_i \cap \sigma| = \tilde{O}(\frac{n_\sigma}{\lceil L/2^i \rceil})$ and $|\hat{Q}_i \cap \sigma'| = \tilde{O}(\frac{n_{\sigma'}}{2^i})$ for every (σ, σ') w.h.p. We already have a selection oracle for each $\delta(\hat{P}_i \cap \sigma, \hat{Q}_i \cap \sigma')$ that w.h.p. runs in time

$$\tilde{O}\left(\left(\frac{n_\sigma}{\lceil L/2^i \rceil}\right)^{2/3} \left(\frac{n_{\sigma'}}{2^i}\right)^{2/3} + n_\sigma + n_{\sigma'}\right) = \tilde{O}\left(\frac{n_\sigma^{2/3} n_{\sigma'}^{2/3}}{L^{2/3}} + n_\sigma + n_{\sigma'}\right).$$

To obtain a selection oracle for $\delta_{\text{light}}(\widehat{P}_i, \widehat{Q}_i)$, we face a problem analogous to selection in a union of multiple sorted arrays, except that the arrays are implicitly given. Frederickson and Johnson [18] gave an algorithm for selection in k sorted arrays of size n running in $O(k \log n)$ time; the algorithm proceeds in $O(\log n)$ rounds, and in each round, it looks up only $O(1)$ elements in each array. In our scenario, each array lookup corresponds to a call to the selection oracle for $\delta(\widehat{P}_i \cap \sigma, \widehat{Q}_i \cap \sigma')$ for a light pair (σ, σ') . Thus, by Frederickson and Johnson’s algorithm, selection in the union $\delta_{\text{light}}(\widehat{P}_i, \widehat{Q}_i)$ can be done in total time

$$\begin{aligned} & \tilde{O} \left(\sum_{\text{light pair } (\sigma, \sigma')} \left(\frac{n_\sigma^{2/3} n_{\sigma'}^{2/3}}{L^{2/3}} + n_\sigma + n_{\sigma'} \right) \right) \\ & \leq \tilde{O} \left(\sum_{\text{light pair } (\sigma, \sigma')} \frac{\Delta^{1/3} (n_\sigma + n_{\sigma'})}{L^{2/3}} + n_\sigma + n_{\sigma'} \right) = \tilde{O} \left(\frac{\Delta^{1/3} n}{L^{2/3}} + n \right), \end{aligned}$$

since $\min\{n_\sigma, n_{\sigma'}\} \leq \Delta$ for each light pair (σ, σ') , and $\sum_{\text{neighboring pair } (\sigma, \sigma')} (n_\sigma + n_{\sigma'}) = O(n)$. Therefore, the total time for interval shrinking with respect to δ_{light} is $\tilde{O}(\Delta^{1/3} n / L^{2/3} + D(n))$ w.h.p.

3.4 Computing Distances with $O(n/\Delta)$ Additive Error

Let H_r be the graph obtained from $G_r(P)$ by contracting the points inside each heavy cell of Ψ_r into a single vertex. Since there are $O(n/\Delta)$ contracted vertices in H_r , we have $d_r(s, p) - k \leq d_{H_r}(s, p) \leq d_r(s, p)$ with $k = O(n/\Delta)$, for every $p \in P$.

It is straightforward to modify Chan and Skrepetos’ decision algorithm to do BFS on this modified graph H_r . Expanding the frontier again reduces to solving Subproblem 5 for pairs of neighboring cells. Note that for two neighboring cells that are both heavy, we can determine whether there is an edge between them by pre-computing the bichromatic closest pair between the points in the neighboring cells (via Voronoi diagrams) and testing r against this closest pair distance – call this a *special* distance. This pre-computation takes $O(n \log n)$ total time.

After performing interval shrinking as described in previous subsection, we apply the bifurcation technique (Lemma 3) to simulate BFS on H_{r^*} . The critical values indeed are all Euclidean distances from light pairs only, except for the $O(n)$ special distances above, but initially before the simulation starts, we can resolve the comparisons with all $O(n)$ special distances by binary search using $O(\log n)$ calls to the decision algorithm in $\tilde{O}(n)$ time.

At the end of the simulation, we will then have computed $d_{H_{r^*}}(s, p)$ for all $p \in P$, which yield estimates to $d_{r^*}(s, p)$ with additive error $O(n/\Delta)$. To summarize:

► **Lemma 6.** *In time $\tilde{O}(\Delta^{1/3} n / L^{2/3} + L^{1/2} n)$, one can compute for every $p \in P$ an estimate \tilde{d}_p with $d_{r^*}(s, p) - k \leq \tilde{d}_p \leq d_{r^*}(s, p)$, where $k = O(n/\Delta)$.*

3.5 Recovering r^*

Finally, we will compute r^* using the estimates obtained in Lemma 6. This final step is done via a dynamic program, and does not require any knowledge of the details of the earlier steps.

► **Lemma 7.** *Given a number k and estimates \tilde{d}_p with $d_{r^*}(s, p) - k \leq \tilde{d}_p \leq d_{r^*}(s, p)$ for all $p \in P$, one can compute r^* in $\tilde{O}(kn)$ time.*

32:12 Faster Algorithms for Reverse Shortest Path in Unit-Disk Graphs

To prove Lemma 7, it is helpful to view the problem as computing a *minimum bottleneck path* from s to t with at most λ links (as mentioned in the introduction), i.e., a path from s to t with at most λ links that minimizes the maximum Euclidean length of the edges along the path.

There is a straightforward dynamic programming solution to this problem: Let $D_i(p)$ denote the minimum bottleneck value over all paths from s to p with i links. We want $r^* = \min_{i \leq \lambda} D_i(t)$. We have the following recursive formula: for each $p \in P$,

$$D_i(p) = \min_{q \in P} \max\{D_{i-1}(q), \delta(p, q)\}, \quad (1)$$

with $D_0(s) = 0$ and $D_0(p) = \infty$ for all $p \in P - \{s\}$ as base cases. Unfortunately, in this dynamic program, the number of table entries is $O(n\lambda)$, which is $O(n^2)$ in the worst case.

We use the given estimates \tilde{d}_p to speed up the dynamic program. The main observation is that it suffices to generate $D_i(p)$ when $i - k \leq \tilde{d}_p \leq i$. The number of such table entries is thus reduced to $O(kn)$.

More precisely, define a subset $P_i = \{p \in P : i - k \leq \tilde{d}_p \leq i\}$ for each i . Since a point belongs to $O(k)$ of these subsets, $\sum_i |P_i| = O(kn)$. For each $p \in P_i$, define

$$\widehat{D}_i(p) = \min_{q \in P_{i-1}} \max\{\widehat{D}_{i-1}(q), \delta(p, q)\}, \quad (2)$$

with $\widehat{D}_0(s) = 0$. (All unspecified $\widehat{D}_i(p)$ values are implicitly set to ∞ .)

We claim that $r^* = \min_{i \leq \lambda} \widehat{D}_i(t)$.

Correctness. It is obvious that $\widehat{D}_i(p) \geq D_i(p)$ for all p and i . Thus, $\min_{i \leq \lambda} \widehat{D}_i(t) \geq \min_{i \leq \lambda} D_i(t) = r^*$.

For the reverse direction, take a shortest path $\langle u_0, u_1, \dots, u_i \rangle$ from s to t in $G_{r^*}(P)$, with $u_0 = s$, $u_i = t$, and $i \leq \lambda$. Note that $d_{r^*}(s, u_j) = j$ for every $j = 0, \dots, i$. So, $j - k \leq \tilde{d}_{u_j} \leq j$, i.e., $u_j \in P_j$ for every $j = 0, \dots, i$. It follows from (2) that $\widehat{D}_0(u_0) = 0$, $\widehat{D}_1(u_1) \leq r^*$, $\widehat{D}_2(u_2) \leq r^*$, \dots , $\widehat{D}_i(u_i) \leq r^*$ (since $\delta(u_0, u_1), \dots, \delta(u_{i-1}, u_i) \leq r^*$). Thus, $\min_{i \leq \lambda} \widehat{D}_i(t) \leq r^*$.

Running time. For an efficient implementation of the dynamic program, we need a data structure for the following variant of nearest neighbor queries:

► **Lemma 8.** *Given a set Q of n points in the plane where each point q has a weight w_q , we can preprocess Q in $O(n \log^2 n)$ time so that given any query point p , we can find $q \in Q$ minimizing $\max\{\delta(p, q), w_q\}$ in $O(\log^2 n)$ time, where $\delta(\cdot, \cdot)$ denotes Euclidean distance.*

Proof. We apply standard multi-level geometric data structuring techniques [2]: Order Q by increasing weights. Split Q into the left half Q_L and the right half Q_R . Store a point location structure for the Voronoi diagram of Q_L [15]. Recursively build data structures for Q_L and for Q_R . The preprocessing time satisfies the recurrence $P(n) = 2P(n/2) + O(n \log n)$.

Given a query point p , we first find its nearest neighbor q_L in Q_L in $O(\log n)$ time.

- Case 1: $\delta(p, q_L) \leq w_{q_L}$. Then $\min_{q \in Q_R} \max\{\delta(p, q), w_q\} \geq w_{q_L} = \max\{\delta(p, q_L), w_{q_L}\}$, and so it suffices to recurse in Q_L .
- Case 2: $\delta(p, q_L) > w_{q_L}$. Then $\min_{q \in Q_L} \max\{\delta(p, q), w_q\} = \delta(p, q_L)$, and so it suffices to recurse in Q_R .

The query time satisfies the recurrence $Q(n) = Q(n/2) + O(\log n)$, and thus we have $Q(n) = O(\log^2 n)$. ◀

Having computed $\widehat{D}_{i-1}(\cdot)$, we can evaluate $\widehat{D}_i(\cdot)$ according to (2) by preprocessing P_{i-1} in the data structure from Lemma 8 (with weights $w_q := \widehat{D}_{i-1}(q)$) and answering $|P_i|$ queries. The running time is $\tilde{O}(|P_{i-1}| + |P_i|)$. Summing over all i yields total running time $\tilde{O}(kn)$. This completes the proof of Lemma 7.

► **Theorem 9.** *The reverse shortest path problem in (unweighted) unit-disk graphs can be solved in $\tilde{O}(n^{9/8})$ time w.h.p. by a (Las Vegas) randomized algorithm.*

Proof. By combining Lemma 6 and Lemma 7, we obtain an algorithm that runs in time

$$\tilde{O}\left(\frac{\Delta^{1/3}n}{L^{2/3}} + L^{1/2}n + \frac{n^2}{\Delta}\right).$$

Choosing $L = n^{1/4}$ and $\Delta = n^{7/8}$ yields the desired result. ◀

References

- 1 Pankaj K. Agarwal, Boris Aronov, Micha Sharir, and Subhash Suri. Selecting distances in the plane. *Algorithmica*, 9(5):495–514, 1993. doi:10.1007/BF01187037.
- 2 Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. AMS Press, 1999.
- 3 Pankaj K. Agarwal, Haim Kaplan, Matthew J. Katz, and Micha Sharir. Segment proximity graphs and nearest neighbor queries amid disjoint segments. In *Proc. 32nd Annual European Symposium on Algorithms (ESA)*, volume 308 of *LIPICs*, pages 7:1–7:20, 2024. doi:10.4230/LIPICs.ESA.2024.7.
- 4 Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. On reverse shortest paths in geometric proximity graphs. *Comput. Geom.*, 117:102053, 2024. Preliminary version in ISAAC 2022. doi:10.1016/J.COMGEO.2023.102053.
- 5 Pankaj K. Agarwal and Micha Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998. doi:10.1145/299917.299918.
- 6 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Trans. Algorithms*, 11(4):29:1–29:29, 2015. Preliminary version in SoCG 2014. doi:10.1145/2700222.
- 7 Sergio Cabello and Miha Ježič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015. doi:10.1016/J.COMGEO.2014.12.003.
- 8 Timothy M. Chan. Geometric applications of a randomized optimization technique. *Discret. Comput. Geom.*, 22(4):547–567, 1999. doi:10.1007/PL00009478.
- 9 Timothy M. Chan. On enumerating and selecting distances. *Int. J. Comput. Geom. Appl.*, 11(3):291–304, 2001. doi:10.1142/S0218195901000511.
- 10 Timothy M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–436, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982853>.
- 11 Timothy M. Chan, Sariel Har-Peled, and Mitchell Jones. Optimal algorithms for geometric centers and depth. *SIAM J. Comput.*, 51(3):627–663, 2022. doi:10.1137/21M1423324.
- 12 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *Proc. 27th International Symposium on Algorithms and Computation (ISAAC)*, volume 64 of *LIPICs*, pages 24:1–24:13, 2016. doi:10.4230/LIPICs.ISAAC.2016.24.
- 13 Timothy M. Chan and Da Wei Zheng. Hopcroft’s problem, \log^* shaving, two-dimensional fractional cascading, and decision trees. *ACM Trans. Algorithms*, 20(3):24, 2024. doi:10.1145/3591357.

- 14 Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987. doi:10.1145/7531.7537.
- 15 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008. URL: <https://www.worldcat.org/oclc/227584184>, doi:10.1007/978-3-540-77974-2.
- 16 Michael B. Dillencourt, David M. Mount, and Nathan S. Netanyahu. A randomized algorithm for slope selection. *Int. J. Comput. Geom. Appl.*, 2(1):1–27, 1992. doi:10.1142/S0218195992000020.
- 17 Alon Efrat, Alon Itai, and Matthew J Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31:1–28, 2001. doi:10.1007/s00453-001-0016-8.
- 18 Greg N. Frederickson and Donald B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *J. Comput. Syst. Sci.*, 24(2):197–208, 1982. doi:10.1016/0022-0000(82)90048-4.
- 19 Greg N. Frederickson and Donald B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM J. Comput.*, 13(1):14–30, 1984. doi:10.1137/0213002.
- 20 Te C. Hu. The maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961.
- 21 Haim Kaplan, Matthew J. Katz, Rachel Saban, and Micha Sharir. The unweighted and weighted reverse shortest path problem for disk graphs. In *Proc. 31st Annual European Symposium on Algorithms (ESA)*, volume 274 of *LIPICs*, pages 67:1–67:14, 2023. doi:10.4230/LIPICs.ESA.2023.67.
- 22 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discret. Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/S00454-020-00243-7.
- 23 Matthew J. Katz, Rachel Saban, and Micha Sharir. Near-linear algorithms for visibility graphs over a 1.5-dimensional terrain. In *Proc. 32nd Annual European Symposium on Algorithms (ESA)*, volume 308 of *LIPICs*, pages 77:1–77:17, 2024. doi:10.4230/LIPICs.ESA.2024.77.
- 24 Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26(5):1384–1408, 1997. doi:10.1137/S0097539794268649.
- 25 Jirí Matoušek. Randomized optimal algorithm for slope selection. *Inf. Process. Lett.*, 39(4):183–187, 1991. doi:10.1016/0020-0190(91)90177-J.
- 26 Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983. doi:10.1145/2157.322410.
- 27 Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discret. Comput. Geom.*, 64(4):1141–1166, 2020. doi:10.1007/S00454-020-00219-7.
- 28 Haitao Wang and Yiming Zhao. Improved algorithms for distance selection and related problems. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPICs*, pages 101:1–101:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICs.ESA.2023.101.
- 29 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *Journal of Computational Geometry*, 14(1):14–47, 2023. Preliminary versions in WADS 2021 and WALCOMM 2022. doi:10.20382/JOCG.V14I1A2.