The Unweighted and Weighted Reverse Shortest Path Problem for Disk Graphs

Haim Kaplan
School of Computer Science, Tel Aviv University, Israel

Matthew J. Katz
Department of Computer Science, Ben Gurion University, Beer-Sheva, Israel

Rachel Saban
Department of Computer Science, Ben Gurion University, Beer-Sheva, Israel

Micha Sharir
School of Computer Science, Tel Aviv University, Israel

Abstract

We study the reverse shortest path problem on disk graphs in the plane. In this problem we consider the proximity graph of a set of n disks in the plane of arbitrary radii: In this graph two disks are connected if the distance between them is at most some threshold parameter r. The case of intersection graphs is a special case with $r = 0$. We give an algorithm that, given a target length k, computes the smallest value of r for which there is a path of length at most k between some given pair of disks in the proximity graph. Our algorithm runs in $O^*(n^{5/4})$ randomized expected time, which improves to $O^*(n^{6/5})$ for unit disk graphs, where all the disks have the same radius.[1] Our technique is robust and can be applied to many variants of the problem. One significant variant is the case of weighted proximity graphs, where edges are assigned real weights equal to the distance between the disks or between their centers, and k is replaced by a target weight w. In other variants, we want to optimize a parameter different from r, such as a scale factor of the radii of the disks.

The main technique for the decision version of the problem (determining whether the graph with a given r has the desired property) is based on efficient implementations of BFS (for the unweighted case) and of Dijkstra's algorithm (for the weighted case), using efficient data structures for maintaining the bichromatic closest pair for certain bicliques and several distance functions. The optimization problem is then solved by combining the resulting decision procedure with enhanced variants of the interval shrinking and bifurcation technique of [4].

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, geometric optimization, disk graphs, BFS, Dijkstra's algorithm, reverse shortest path

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.67

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2307.14663

Funding *Haim Kaplan*: Work partially supported by Grant 1595/19 from the Israel Science Foundation and by the Blavatnik Research Foundation.
Matthew J. Katz: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Micha Sharir: Work partially supported by Grant 260/18 from the Israel Science Foundation.

1 Introduction

In this paper we study the *reverse shortest path problem* (RSP for short) on graphs defined by disks in the plane.

[1] In this paper the $O^*(\cdot)$ notation hides subpolynomial factors.

In the simplest variant of this problem, we are given a set P of n points in the plane, two designated points $s, t \in P$, a real parameter $r > 0$ and an integer $k < n$. Define G_r to be the graph (P, E_r), where the edges of E_r are all the pairs (p, q) such that $\|p - q\| \leq 2r$. This is also the intersection graph of the disks of radius r centered at the points of P. In the decision version of the RSP problem we want to determine whether G_r contains a path from s to t with at most k edges. In the optimization version, which is the reverse shortest path problem itself, we wish to find the smallest value r^* for which G_{r^*} has this property. Both versions (decision and optimization) of this problem have received considerable attention during the past decade [5, 7, 14].

Our contributions. We give an algorithm for this problem that runs in $O^*(n^{6/5})$ randomized expected time (where the $O^*(\cdot)$ notation hides subpolynomial factors). This improves the recent $O^*(n^{5/4})$-time solution of Wang and Zhao [14]. In fact we study the RSP problem in a much more general context that involves a variety of intersection or proximity graphs on a finite set of arbitrary disks in the plane, for which nontrivial bounds were not known prior to this work.

We first consider unweighted disk graphs in Section 3. In this setup, we have a set $\mathcal{D}$ of n disks in the plane of arbitrary radii. Each disk $D \in \mathcal{D}$ is specified by its center c_D and radius ρ_D. For a given parameter $r \geq 0$ we define the *proximity graph* G_r by adding an edge between disks D and D' if the distance between them, $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}$, is at most r. The case $r = 0$ is of special interest and gives the intersection graph of the disks.[2]

For the decision version of this RSP problem we obtain an algorithm that runs in $O(n \log^4 n)$ time, and for the optimization version an algorithm that runs in $O^*(n^{5/4})$ randomized expected time. Our technique generalizes to other versions of the optimization problem. For example, we can consider the intersection graph of the disks ($r = 0$) and ask for the smallest scaling factor of the radii of all disks, either by a common additive term or by a common multiplicative factor, that would make the graph contain a path of at most k edges between a designated pair of source and target disks D_s and D_t.

In Section 4 we generalize our results further to weighted versions of the proximity graph G_r. We consider two natural weight functions for the edges. The first sets the weight of an edge (D, D') to be the distance $\|c_D - c_{D'}\|$ between the centers of the disks, and the second sets the weight to be the distance $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}$ between the disks. We solve the decision problem on such weighted disk graphs, in which we want to determine whether the shortest path in G_r from D_s to D_t is of length at most w, for some specified real threshold w, by a careful implementation of Dijkstra's algorithm (using a dynamic bichromatic closest pair structure, see below) in $O(n \log^4 n)$ or $O(n \log^6 n)$ time, depending on the type of edge weights. The optimization RSP problem is then solved in $O^*(n^{5/4})$ randomized expected time. For weighted unit disk graphs we still get the better bound of $O^*(n^{6/5})$ time for the optimization problem.

Our decision algorithms rely on rather complex dynamic data structures (see below), which should be avoided, if possible, from a practical point of view. Indeed, for unit disk graphs, there exist simpler and slightly more efficient implementations of BFS and Dijkstra's algorithm, in the unweighted and weighted cases, respectively [5, 7, 13]. However, as explained in the remarks following Theorems 2 and 7, we cannot use them in conjunction with our optimization technique, for certain technical reasons. We thus present in the full version of this paper alternative implementations, based on the known grid-based techniques, which satisfy our requirements and are arguably somewhat simpler.

[2] One technical difference is that when considering proximity graphs, it is customary, although not obligatory, to assume that the disks are pairwise disjoint, which is certainly not an assumption that one would make for intersection graphs.

Our techniques. We achieve our results by carefully combining three main technical ingredients. The first is an efficient "serial" implementation of parametric search, using what we call *interval shrinking* and *bifurcation* procedures. This technique was first used by Ben Avraham et al. [4] for solving problems involving the discrete and semi-discrete Fréchet distance with shortcuts. Here we apply a somewhat modified variant of it in the rather different context of our RSP problem, exposing its potential to be useful for a wide range of other problems as well.

We remark, that using this technique requires that the decision procedure access the parameter r to be optimized via comparisons only, whose outcome depends on the relation between the optimal r^* and certain critical values (which in our case turn out to be additively weighted inter-point distances between the centers of the disks), on which we can apply the interval shrinking procedure in an efficient manner. We review this technique in Section 2.

The second ingredient is a dynamic nearest neighbor and a dynamic bichromatic closest pair data structures for additively weighted Euclidean distances. Such structures with polylogarithmic time per update and access were recently developed by Kaplan et al. [8] and Liu [11].

The third ingredient that we use for the weighted versions of the problem is a technique that combines nearest neighbor data structures for two different distance functions. Specifically, given a (dynamic) nearest neighbor data structure for a distance function d_1, and a (dynamic) nearest neighbor data structure for a distance function d_2, we show how we can get a (dynamic) data structure that can answer constrained nearest neighbor queries of the form: find the closest point to a query q according to the distance function d_1 among all points whose distance to q according to d_2 is at most some threshold r (which is part of the query).

Previous work. The decision problem in the unweighted variants can be solved by running a BFS from s (or from D_s) in the underlying graph. Similarly, the decision problem in the weighted variants can be solved by running Dijkstra's shortest-path algorithm in the graph. However, the challenge is to do it efficiently, since the graph might have up to a quadratic number of edges. For unit-disk graphs, Cabello and Jejčič [5] presented an $O(n \log n)$ implementation of BFS, and subsequently Chan and Skrepetos [7] presented an alternative $O(n)$ implementation (after pre-sorting the points by their x- and y-coordinates). Moreover, Cabello and Jejčič [5] also described an $O(n^{1+\varepsilon})$ implementation of Dijkstra's algorithm for weighted unit-disk graphs, which was followed by a more efficient $O(n \log^2 n)$ implementation described by Wang and Xue [13]; see also [8].

The RSP problem in the context of unweighted unit-disk graphs was posed by Cabello and Jejčič [5], who observed that it can be solved conceptually easily in $O^*(n^{4/3})$ time, by running a binary search through the $O(n^2)$ inter-point distances (using an efficient distance selection algorithm). Recently, Wang and Zhao [14] managed to improve this bound, obtaining an algorithm that solves the problem in $O^*(n^{5/4})$ time. In the context of weighted unit-disk graphs, the situation is similar. The RSP problem can be solved easily in $O^*(n^{4/3})$ time, but Wang and Zhao [15] were able to obtain an improved $O^*(n^{5/4})$-time solution for that version too. [3]

As far as we know, both the decision and optimization problems have not been studied in the context of general disk graphs.

[3] The $O^*(n^{6/5})$ bound for the RSP problem in both unweighted and weighted unit disk graphs was already claimed in an unpublished manuscript [9], which appeared shortly after the first RSP paper of Wang and Zhao. However, this manuscript overlooks an issue that may arise when using an off-the-shelf decision problem, see the full version of this paper.

2 Preliminaries

In this section we provide necessary background on the serial parametric search technique of Ben Avraham et al. [4].

In its basic form, the technique is applicable when the threshold parameter r^* is the distance between a pair of input points. There are $O(n^2)$ such distances, and the most naïve algorithm simply finds r^* by running a binary search through them, guided by the decision procedure at each comparison. A basic improvement is to implement the binary search using an efficient procedure for distance selection, such as the one in [2] or [10], which runs in $O^*(n^{4/3})$ time. Up to an additional logarithmic factor, this dominates the cost of the whole procedure (assuming that the decision procedure is more efficient; as we show, this is indeed the case in all the RSP problems studied in this paper).

The technique of [4] is a combination of two subprocedures, referred to as the *interval shrinking* and the *bifurcation* procedures. The interval shrinking procedure receives an integer parameter $L \ll \binom{n}{2}$, and computes an interval $I \subset \mathbb{R}$ that contains r^* and at most L *critical values*, namely inter-point distances. As shown in [4], this can be done in $O^*(n^{4/3}/L^{1/3})$ expected time.

We then run the bifurcation procedure, which simulates the execution of the decision procedure at the (unknown) threshold r^*, as in the standard parametric search technique [12]. When the simulation reaches a comparison of r^* with some concrete value r, we know the answer to the comparison when r lies outside I. However, when $r \in I$ we bifurcate, following both possibilities $r^* > r$ and $r^* < r$ (the case $r^* = r$ will be handled too; see below). This produces a *bifurcation tree* T, which we expand until we either collect sufficiently many bifurcations, or until we reach a sufficiently large uniform depth of T. In either case we stop this simulation phase, resolve all collected comparisons by a binary search through them, using the (unsimulated) decision procedure to guide the search, and start a new bifurcation phase from the unique leaf of T whose associated (shrunk) interval of critical values contains r^*. The binary search will also identify r^* when it is one of the critical values through which it searches, and then terminate the entire procedure right away. In the worst case this will happen by the time when the entire decision procedure has been simulated.

We comment that this method is viable when the decision procedure is not known to have a parallel version of small depth, which is required in the standard parametric search technique. If such a parallel version were available, we could apply standard parametric search, and obtain a significantly faster algorithm. The RSP problem seems to be inherently sequential, as it seeks a path in a graph, and is indeed amenable to the technique of [4].

As shown in [4], the bifurcation procedure can be implemented to run in $O^*(L^{1/2}D(n))$ time, where $D(n)$ is the cost of the decision procedure. A suitable choice of L yields an overall (randomized expected) running time $O^*(n^{6/5})$, for (suitable) decision procedures that run in nearly linear time, as do the decision procedures for all the variants of the RSP problem considered in this paper.

3 Reverse shortest paths for unweighted disk graphs

Here we are given a set $\mathcal{D}$ of n disks in the plane, of arbitrary radii, each parameterized by its center c_D and radius ρ_D. We consider the *intersection graph* $G^\times = (\mathcal{D}, E)$, where the edges of $G^\times$ are the intersecting pairs of disks. Formally, E consists of all pairs (D, D') for which $\|c_D - c_{D'}\| \leq \rho_D + \rho_{D'}$.

In the decision problem, we are given two designated disks D_s and D_t, and an integer parameter k, and the goal is to determine whether $G^\times$ contains a path of at most k edges from D_s to D_t.

In the optimization problem we scale the radii of the disks (without changing their centers), either by an additive term or by a multiplicative factor, and seek the smallest scaling parameter that makes $G^\times$ have the desired s-t path.

Towards the end of the section, we consider the special and important case of unit disk graphs, for which we obtain a better bound.

3.1 The decision procedure

We run BFS on $G^\times$ from D_s. Suppose that we have already discovered all disks at some level i of the BFS. We expand the BFS to level $i+1$ as follows. We consider each disk D at level i in turn, and look for its nearest neighbor among the disks that have not yet been discovered. To do so, we maintain a dynamic additively-weighted Voronoi diagram $\mathrm{Vor}(\mathcal{U})$ of the set $\mathcal{U}$ of all the disks that the BFS has not yet reached, where the additive weight of a disk D is $-\rho_D$. Initially, we are at level 0 of the BFS, which includes only D_s, and we set $\mathcal{U} = \mathcal{D} \setminus \{D_s\}$.

We search for the nearest neighbor D' of D in $\mathrm{Vor}(\mathcal{U})$. If the weighted distance between D and D' is at most ρ_D, that is, if $\|c_D - c_{D'}\| - \rho_{D'} \leq \rho_D$, we conclude that $\|c_D - c_{D'}\| \leq \rho_D + \rho_{D'}$, so we add D' to level $i+1$, delete it from $\mathcal{U}$, and query the updated Voronoi diagram for the new nearest neighbor of D again. We continue querying the updated $\mathrm{Vor}(\mathcal{U})$ with D until the distance between D and its nearest neighbor is larger than ρ_D. When this happens we replace D by the next disk at level i and repeat this process. When we finish processing in this manner all disks of level i, we have discovered all disks at level $i+1$, and we move on to level $i+1$.

Consider a sequence of queries to the Voronoi diagram with a disk D at some level of the BFS. We can charge each of these queries but the last, to a new disk that we add, following this query, to the BFS tree. Therefore the running time of this decision procedure is dominated by the cost of $O(n)$ queries and n deletions from $\mathrm{Vor}(\mathcal{U})$. The most efficient implementation of such a structure, with running time $O(n \log^4 n)$, is due to Liu [11]; see also the earlier study [8], with a worse polylogarithmic factor.

In summary, we have shown:

▶ **Theorem 1.** *Given $\mathcal{D}$, D_s, D_t, and k as above, we can determine whether there exists a path of at most k edges from D_s to D_t in the intersection graph $G^\times$ associated with $\mathcal{D}$, in $O(n \log^4 n)$ time.*

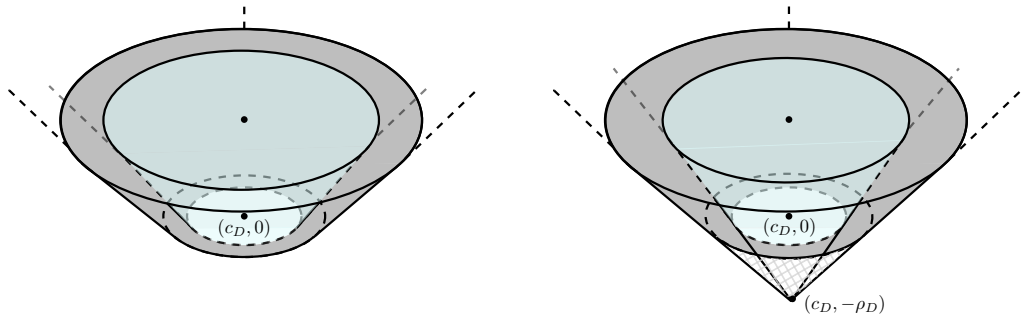
3.2 The optimization procedure

Each disk $D \in \mathcal{D}$ is assigned the radius $\rho_D + \alpha$, for some common additive parameter α, and we want to find the minimum value α^* of α for which the intersection graph of the modified disks, now denoted by $G^\times_\alpha$, has the desired s-t path. In principle α could also be negative, as long as no radius becomes negative, but for simplicity we only consider the case $\alpha > 0$.

We simulate the decision procedure at the unknown optimal value α^* by using a bifurcation procedure. Before starting the simulation, though, we perform an interval-shrinking step, as described in the introduction.

Interval shrinking. Recall that this step, as introduced in [4], receives an integer threshold parameter L and produces an interval $I_0 \subset \mathbb{R}$ that contains α^* and at most L other critical values, where a value α is critical if the outcome of a comparison changes as we go past α. In the original formulation, the critical values were inter-point distances in the plane, and the resulting algorithm ran in $O^*(n^{4/3}/L^{1/3})$ randomized expected time.

Here the setup is different. The comparisons that the decision procedure performs are tests whether expressions of the form $\|c_D - c_{D'}\| - \rho_D - \rho_{D'} - 2\alpha$ are positive or negative. The critical values of the parameter α are thus of the form $\frac{1}{2}\left(\|c_D - c_{D'}\| - \rho_D - \rho_{D'}\right)$. The original mechanism of [4] is based on distance selection. We need a variant in which the basic step is to bound the number of these new critical values in a given interval (α_1, α_2). We turn this problem into a range searching problem, where the disks of $\mathcal{D}$ serve as both data and query objects. Specifically, each disk D is mapped to the point (c_D, ρ_D) in $\mathbb{R}^3$, and also to the range $\sigma_D = \{D' \mid 2\alpha_1 \leq \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq 2\alpha_2\}$, which is a conical shell in 3-space (see Figure 1(a)). Note that the ranges have three degrees of freedom, and that the problem is symmetric, so that ranges can be represented as points in $\mathbb{R}^3$ and points as ranges. In other words, we have a symmetric batched range searching problem in $\mathbb{R}^3$, involving n points and n semi-algebraic ranges. Using standard, cutting-based decomposition techniques in $\mathbb{R}^3$, such as in [1, 3], we can implement the range searching step to run in randomized expected time $O^*(n^{3/2})$. Combining this with parametric search, as in the standard distance selection procedure [2], we can implement the distance selection procedure to also run in randomized expected time $O^*(n^{3/2})$.



(a) (b)

Figure 1 The range σ_D (in grey) when scaling by an additive term (a) and by a multiplicative factor (b). The inner and outer radii of the annulus centered at $(c_D, 0)$ are $\rho_D + 2\alpha_1$ and $\rho_D + 2\alpha_2$, respectively, in (a) and $\rho_D\lambda_1$ and $\rho_D\lambda_2$, respectively, in (b).

Extending the machinery in [4], we can convert this distance selection technique to an interval-shrinking procedure that receives a parameter $L \ll \binom{n}{2}$ and yields an interval $I = (\alpha_1, \alpha_2)$ that contains the optimum value α^* and at most L critical values. A suitable modification of the analysis in [4] shows that the algorithm runs in $O^*(n^{3/2}/L^{1/2})$ randomized expected time.

Bifurcation. We now present the basic bifurcation procedure. This procedure, sometimes with a few enhancements and modifications, is used in all our algorithms for the various variants of the RSP problem. We refer the reader to [4] where a similar procedure has been used.

Our simulation proceeds in *phases*, where in each phase we construct a bifurcation tree T that represents some portion of the execution of the BFS, simultaneously for all values of α in some interval I. Initially $I = I_0$, but it will keep on shrinking as the simulation proceeds. Each node b of T is associated with an interval $I^b = (\alpha_1, \alpha_2) \subseteq I$, such that, up to the state of simulation represented by b, the BFS proceeds in an identical manner for all values $\alpha \in I^b$. We continue to simulate the BFS at b. At each comparison of (the unknown) α^* with some concrete value α, we either resolve the comparison in a unique manner when $\alpha \notin I^b$, or

else bifurcate, meaning that we create two children u and v of b, assign $I^u := (\alpha_1, \alpha)$ and $I^v := (\alpha, \alpha_2)$, and continue to expand T at u and at v, at each of which we know the outcome of the above procedure (the possibility that $\alpha^* = \alpha$ will be tested later).

For each node b, let y_b denote the amount of work performed at b so far by the simulation (including comparisons that were fully resolved), and let y_b^- denote the sum of the quantities y_a over all proper ancestors a of b, *excluding the root*. We refer to the cost at the root as the *initialization cost* of the tree, and denote it as $C_0(T)$.

We stop the expansion of T at a node b when $y_b^- + y_b = Y$, where Y is a threshold parameter that will be determined later. That is, we stop the simulation at node b as soon as y_b^- plus the work done so far at b becomes Y. We refer to such nodes b as *incomplete leaves* of T. We then continue the expansion of T at other nodes.

A subtle issue, that we will address later in more detail, is that when we back up from an incomplete node b to explore other branches of T, we need to restore the state of execution at the suitable ancestors of b. See below for details.

We stop the entire construction of T as soon as one of the following two conditions occurs:

(i) We collect X bifurcations, for another threshold parameter X that will be determined later.

(ii) All the leaves of T are incomplete.

When this happens, we take all the (at most X) critical values of the bifurcations at the inner nodes of T, and run a binary search through them, using the unsimulated decision procedure to guide the search. This takes $O(D(n) \log n)$ time, and yields the leaf w whose range I^w contains r^*. It is also possible that the binary search will detect that one of the critical values it searches through is r^* itself. In this case the entire procedure is terminated, and r^* is output. Otherwise, this ends a phase of the simulation. We start a new phase (if the simulation has not already ended) with w as the root of the tree, and with I^w as the critical interval. Note that this will cause the work already done at w to be repeated, and it is possible that the cost of this work is much larger than Y. However, by charging the incomplete work at w to the initialization cost $C_0(T')$ of the next tree T', we at most double this cost, so this will not affect the overall asymptotic bound on the performance of the procedure; see below for details.

Restoring the execution state. The issue of restoring the state at nodes we back up to, as mentioned earlier, is more acute in our specific application, because part of this state includes the Voronoi diagram that our procedure maintains dynamically. Although there are persistence-based techniques that can efficiently maintain all versions of the diagram, they are fairly involved, and we opt not to use them. Instead, we use the following simple approach, which also takes care of all aspects of restoring the state.

Specifically, we expand the bifurcation tree in a depth-first manner, so that at each node we first recursively construct the subtree of its left child and only then the subtree of its right child. We thus first expand the leftmost path of T, and slowly proceed to the right, backing from a node to its parent, and then proceed to the right child, or further up to the grandparent. When we back up from a node w to its parent v, we simply undo the operations performed at w, including the updates that were done to the diagram, in reverse order, replacing each deletion of a disk by the corresponding reinsertion. This requires us to maintain a log of the updates performed at each node, as well as a log of the other operations, but is otherwise a reasonably simple procedure, which does not affect the asymptotic running time and storage bounds.

3.3 Analysis

By construction, a single phase produces a tree T that has at most X binary nodes, and the cost of producing each path of T is at most Y, ignoring the initialization cost $C_0(T)$. This is easily seen to imply that the cost of generating T is $O(XY + C_0(T))$. Indeed, the cost at each node of T, other than the root, is certainly at most Y, and there are $O(X)$ such nodes The cost of the subsequent binary search through the critical values is $O(D(n) \log n)$. Hence the overall cost of a single phase is $O(XY + C_0(T) + D(n) \log n)$. The number of phases is estimated as follows. If the phase terminates because of Condition (i), it has discovered X critical values among those in the current critical interval, which are outside the new critical interval I^w. Hence the number of such phases is at most L/X.[4]

A phase that terminates because of Condition (ii) consumes Y of the total cost of the decision procedure: when we pass to the next phase we follow a single path of the current T, but all paths use at least Y of the cost, excluding the work done at the root. Note that, by construction, the part of the simulation performed along a path of the tree in one phase, excluding the work performed at the root, is disjoint from the part performed along a path of the tree in a different phase. This is easily seen to imply that the number of phases of type (ii) is at most $D(n)/Y$.

The sum $\sum_T C_0(T)$ of the initialization costs of all the trees is at most $D(n)$, because these initializations perform pairwise disjoint portions of the decision procedure. (Observe that execution at a root is always completed, no matter how expensive it is.)

We set our parameters so that $\frac{L}{X} = \frac{D(n)}{Y}$ and $XY = D(n) \log n$. That is, we choose $X = L^{1/2} \log^{1/2} n$ and $Y = D(n) \log^{1/2} n / L^{1/2}$, and obtain that the overall cost of the simulation is $O(L^{1/2} D(n) \log^{1/2} n)$ (this clearly also subsumes the cost $\sum_T C_0(T)$).

In total, the cost of the optimization procedure is

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

We balance these two terms by choosing $L = n^{1/2}$, and obtain a total cost of $O^*(n^{5/4})$. This result is part of the summary in Theorem 2 (which also includes other variants of the problem), given below.

3.4 Other variants and unit disks

Scaling by a multiplicative factor. Consider first the case of intersection graphs where the radii are scaled by a common multiplicative factor $\lambda > 0$. In this case the critical value induced by a pair of disks D, D' satisfies $\|c_D - c_{D'}\| - \lambda\rho_D - \lambda\rho_{D'} = 0$, or $\lambda = \dfrac{\|c_D - c_{D'}\|}{\rho_D + \rho_{D'}}$. As above, the algorithm requires a procedure that computes the number of critical values in an interval (λ_1, λ_2). So we convert this task to batched range searching in three dimensions, where the ranges are now

$$\sigma_D = \{D' \mid \lambda_1(\rho_D + \rho_{D'}) \le \|c_D - c_{D'}\| \le \lambda_2(\rho_D + \rho_{D'})\}.$$

[4] This is a rather weak aspect of the analysis. For the bound L/X to materialize, the X critical values of each phase must form a prefix or a suffix of the sequence of critical values in the current interval I. In general, when these critical values are more uniformly spread within I, the number of such phases should be much smaller. It is a challenging open problem to turn this intuition into an improved procedure, if possible.

These too are conical shells, but here the bounding cones have a common apex and different opening angles (see Figure 1(b)). Other than this new type of ranges, the preceding machinery proceeds verbatim. The interval shrinking runs in $O^*(n^{3/2}/L^{1/2})$ expected time and the bifurcation procedure runs in $O^*(L^{1/2}n)$. With a proper choice of L, the overall cost is $O^*(n^{5/4})$ expected time.

Proximity graphs. Next, consider the case of proximity graphs. In this case, we assume that the disks of $\mathcal{D}$ are pairwise disjoint, and we are given an additional parameter $r > 0$. (For $r = 0$ we get the intersection graph $G^\times$.) The set of edges of the proximity graph G_r consists of all pairs of disks (D, D') for which $\mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq r$. In the RSP problem for proximity graphs, we seek the smallest value of r for which G_r has the desired s-t path. It is easy to see that this problem can be reduced to the (additive version of the) corresponding problem for intersection graphs, by simply adding $r/2$ to the radius of each of the disks. In the optimization procedure, the critical values are of the form $\|c_D - c_{D'}\| - \rho_D - \rho_{D'}$. Thus, except for the factor $\frac{1}{2}$ used earlier, the procedure is essentially identical to the earlier one.

Unit disks. Finally, consider the special case of unit disks, i.e., where all the radii are equal, and consider the intersection graph of the disks. In this case, we get a better bound, since the critical values are merely (one half of the) inter-point distances, and hence the interval shrinking step can be performed in $O^*(n^{4/3}/L^{1/3})$ randomized expected time, as in [4]. Modifying the expression for the overall running time accordingly, we get $O^*\left(\frac{n^{4/3}}{L^{1/3}} + L^{1/2}n\right)$ randomized expected time. We now balance these two terms by choosing $L = n^{2/5}$, and obtain a total cost of $O^*(n^{6/5})$. In summary, we have shown:

▶ **Theorem 2.** *The reverse shortest path problem for unweighted intersection or proximity graphs of arbitrary disks in the plane can be solved in $O^*(n^{5/4})$ randomized expected time. This bound applies to all the variants of the problem listed above. In the case of unit disks, where all radii are equal, the problem can be solved in $O^*(n^{6/5})$ randomized expected time.*

Remark. In the case of unit disks, the decision procedure itself can be implemented to run faster than $O(n \log^4 n)$. Chan and Skrepetos [7] even present an algorithm that runs in linear time (after a preliminary sorting step); see also [5]. However, the critical values produced by their procedure are not all inter-point distances, which affects the running time of the optimization procedure. To avoid the use of complex dynamic data structures, we modify the algorithm of Chan and Skrepetos, so that it can be combined with the optimization procedure, see the full version of this paper.

4 Reverse shortest paths for weighted disk graphs

Recall that, for a set $\mathcal{D}$ of n disks in the plane and an additional parameter $r \geq 0$, the proximity graph G_r of $\mathcal{D}$ has an edge between every pair of disks such that

$$\mathsf{dist}(D, D') := \|c_D - c_{D'}\| - \rho_D - \rho_{D'} \leq r. \tag{1}$$

The intersection graph $G^\times$ of $\mathcal{D}$ is the proximity graph of $\mathcal{D}$ for $r = 0$, except that in proximity graphs we usually assume that the disks are pairwise disjoint.

We next assign weights to the edges. There are two natural choices for these weights. One is to define the weight of an edge (D, D') as the distance $\|c_D - c_{D'}\|$ between the centers. The other is to define the weight of (D, D') to be $\mathsf{dist}(D, D')$ if $\mathsf{dist}(D, D') \geq 0$ and 0 otherwise (that is 0 if the disks intersect). Note that for intersection graphs only the first choice gives a reasonable weight function.

The case of weighted unit disk graphs is a special case of the problem for intersection graphs $G^{\times}$ with weights equal to the distances between the centers. In this case we have $\|c_D - c_{D'}\| = \mathsf{dist}(D, D') + 2$.

4.1 The decision procedure

We are given two disks D_s, D_t of $\mathcal{D}$, and the proximity graph G_r of $\mathcal{D}$ for some $r \geq 0$, weighted by one of the weight functions above. We want to compute the (length of the) shortest path $\pi(D_s, D_t)$ in G from D_s to D_t. We solve this by a clever implementation of Dijkstra's algorithm. The required sophistication of the implementation depends on the type of the weight function we use. We start with the simpler case in which the weight of (D, D') is $\mathsf{dist}(D, D')$ (or 0 if the disks intersect). Specifically, the same function that defines the graph via the threshold r (Equation (1)) is also used to define the weights.

The high-level approach is similar to that proposed by Cabello and Jejčič [5], although their original algorithm was given only for intersection graphs of unit disks. We briefly recall this technique, adapted to our context.

We maintain a decomposition of $\mathcal{D}$ into three disjoint subsets $\mathcal{R}$, $\mathcal{K}$ and $\mathcal{U}$, where $\mathcal{R} \cup \mathcal{K}$ is the set of disks D for which we already have the correct distance label $\delta(D)$ (the length of the shortest path from D_s), and $\mathcal{U}$ is the remainder of $\mathcal{D}$, consisting of disks whose distance labels have not yet been determined. $\mathcal{R}$ (resp., $\mathcal{K}$) is the set of *active* (resp., *dead*) disks of $\mathcal{R} \cup \mathcal{K}$, meaning that the disks of $\mathcal{R}$ still have outgoing edges in G_r to disks of $\mathcal{U}$, while disks of $\mathcal{K}$ have no such edges. Initially, $\mathcal{R} = \{D_s\}$, $\mathcal{K} = \emptyset$, and $\mathcal{U} = \mathcal{D} \setminus \{D_s\}$.

A single step of our implementation of Dijkstra's algorithm, a so-called *Dijkstra step*, picks the closest pair (D, D') in $\mathcal{R} \times \mathcal{U}$, where the modified distance between D and D' is defined as

$$\mathsf{d}(D, D') = \delta(D) + \mathsf{dist}(D, D'). \tag{2}$$

Then we need to verify that (D, D') is indeed an edge of G_r. If this is not the case, i.e., $\mathsf{dist}(D, D') > r$, we move D from $\mathcal{R}$ to $\mathcal{K}$, and never process D again. This action is justified by the following variant of [5, Lemma 6]:

▶ **Lemma 3.** *Let D and D' be as above, and assume that $\mathsf{dist}(D, D') > r$. Then $\mathsf{dist}(D, D'') > r$ for every disk $D'' \in \mathcal{U}$.*

Proof. By assumption, and since (D, D') is the closest pair in $\mathcal{R} \times \mathcal{U}$, we have, for each $D'' \in \mathcal{U}$, $\delta(D) + r < \delta(D) + \mathsf{dist}(D, D') \leq \delta(D) + \mathsf{dist}(D, D'')$, so $\mathsf{dist}(D, D'') > r$, as asserted. ◀

Note that Lemma 3 also applies to weighted unit disk graphs (with weight $\|c_D - c_{D'}\|$ for edge (D, D').)

Assume then that $\mathsf{dist}(D, D') \leq r$. We move D' from $\mathcal{U}$ to $\mathcal{R}$, assign to it the distance label $\delta(D') = \delta(D) + \mathsf{dist}(D, D')$, and set $\mathsf{prev}(D') = D$, namely D is the disk preceding D' along the shortest path to D'. We then repeat the entire step with the new sets $\mathcal{R}$ and $\mathcal{U}$, and continue until $\mathcal{R}$ empties out. Upon termination, $\mathcal{U}$ is the set of disks that are unreachable from D_s in G_r, and $\mathcal{K}$ is the set of reachable disks, each with its correct distance label and its predecessor.

The correctness of this procedure is argued exactly as in [5], even though the distance function is different. An efficient implementation is obtained by applying the efficient dynamic bichromatic closest-pair data structure of Kaplan et al. [8] and of Liu [11] to $\mathcal{R} \times \mathcal{U}$, under the distance function given in (2) and under deletions from $\mathcal{U}$ and insertions into and deletions from $\mathcal{R}$. The technique requires the following two properties.

- **(i)** The Voronoi diagram $\mathrm{Vor}_1(\mathcal{U})$ of $\mathcal{U}$, under the distance function $\mathsf{d}_1(q, D') = \|q - c_{D'}\| - \rho_{D'}$, for $D' \in \mathcal{U}$, where $c_{D'}$ and $\rho_{D'}$ are the respective center and radius of D', has linear complexity.
- **(ii)** The Voronoi diagram $\mathrm{Vor}_2(\mathcal{R})$ of $\mathcal{R}$, under the distance function $\mathsf{d}_2(q, D) = \delta(D) + \|q - c_D\| - \rho_D$, for $D \in \mathcal{R}$, also has linear complexity.

Both properties indeed hold, as each diagram is an additively-weighted Voronoi diagram of the set of centers of the disks of $\mathcal{U}$ or of $\mathcal{R}$. Both diagrams need to be maintained dynamically, and the techniques of [8, 11] do that, with a polylogarithmic cost for each update and query operation. In the more efficient implementation of [11], the amortized cost of an update is $O(\log^4 n)$.

In summary, we have shown:

▶ **Theorem 4.** *The shortest-path tree from some starting disk in the proximity graph of n disks (of arbitrary radii) in the plane, under the weights of inter-disk distances, can be constructed in $O(n \log^4 n)$ time.*

Next we address the more challenging weighted case when the edge weights are the distances between the centers.

The edge weights are the distances between the centers. In this case we need to maintain a dynamic bichromatic closest pair structure under the distance $\lambda(D, D') = \delta(D) + \|c_D - c_{D'}\|$, for $D \in \mathcal{R}$ and $D' \in \mathcal{U}$. But then Lemma 3 does not hold anymore, because it is then possible that the closest pair $(D, D') \in \mathcal{R} \times \mathcal{U}$ satisfies $\mathsf{dist}(D, D') > r$ but there could be other pairs (D, D'') with $\mathsf{dist}(D, D'') \leq r$, so D cannot be removed from $\mathcal{R}$ yet.

To overcome this difficulty we obtain a *bichromatic closest pair* (BCP for short) data structure by applying a black-box reduction of Chan [6] to two novel nearest-neighbor data structures that we now describe.

Our new data structures find for $D \in \mathcal{R}$ a nearest neighbor $D' \in \mathcal{U}$ according to the distance function λ but only among disks D' such that $dist(D, D') \leq r$, and similarly for $D \in \mathcal{U}$.

We then run the Dijkstra steps, using the BCP data structure, as long as there are still neighbors in $\mathcal{R} \times \mathcal{U}$. Upon termination, $\mathcal{U}$ is the set of disks that are unreachable from D_s in G_r, and $\mathcal{R}$ is the set of reachable disks, each with its correct distance label and its predecessor.

Our nearest-neighbor data structure consists of two balanced search trees $T_{\mathcal{U}}$ and $T_{\mathcal{R}}$. The tree $T_{\mathcal{U}}$ ($T_{\mathcal{R}}$ is handled in a similar manner; see below) stores the disks $D' \in \mathcal{U}$ at its leaves, sorted in increasing order of the values $\rho_{D'}$. For each node v of $T_{\mathcal{U}}$, we maintain an additively-weighted Voronoi diagram $\mathrm{Vor}_1(\mathcal{U}_v)$ on the set $\mathcal{U}_v$ of the disks of $\mathcal{U}$ stored at the leaves of the subtree rooted at v, where the weight of a disk D' is $-\rho_{D'}$. We also maintain a second standard (unweighted) Voronoi diagram $\mathrm{Vor}_2(\mathcal{U}_v)$ for $\mathcal{U}_v$.

Querying $T_{\mathcal{U}}$ with a disk $D \in \mathcal{R}$ is performed as follows. We find the leftmost leaf w_0 of $T_{\mathcal{U}}$ whose disk D'_0 satisfies $\mathsf{dist}(D, D'_0) = \|c_D - c_{D'_0}\| - \rho_D - \rho_{D'_0} \leq r$. To find w_0, we search in $T_{\mathcal{U}}$ starting at the root. At each node u that we reach, with a left child v and a right child

w, we search with D in $\mathrm{Vor}_1(\mathcal{U}_v)$, and obtain its nearest neighbor D' in the corresponding subset of disks. If $\|c_D - c_{D'}\| - \rho_{D'} \leq \rho_D + r$, or, equivalently, $\mathsf{dist}(D, D') \leq r$, then we continue the search at the left child v. Otherwise we continue the search with the right child w. At the root we first search in $\mathrm{Vor}_1(\mathcal{U}_{\mathrm{root}} = \mathcal{U})$. If the nearest neighbor D' satisfies $\|c_D - c_{D'}\| - \rho_{D'} > \rho_D + r$, that is, $\mathsf{dist}(D, D') > r$, we conclude that D has no neighbor in $\mathcal{U}$.

Let w_0 be the leaf that the search reaches. Note that if a disk D' is stored at a leaf to the left of w_0 then, by construction, $\mathsf{dist}(D, D') > r$. That is, only disks stored to the right of w_0, including w_0, need to be considered.

The search for w_0 provides us with a representation of the set $\mathcal{U}^+_{w_0}$ of the disks to the right of w_0 as the union of $O(\log n)$ pairwise disjoint canonical sets, each being the set of disks stored at the root of some subtree of $T_{\mathcal{U}}$. We query with D in each of the $O(\log n)$ diagrams $\mathrm{Vor}_2(\mathcal{U}_v)$ that correspond to these subtrees, and return the disk that is nearest to D, i.e., with the minimum distance between their centers, among all the resulting nearest neighbors. The correctness of this procedure is a consequence of the following lemma.

▶ **Lemma 5.** *In the above procedure, the output disk D' satisfies $\mathsf{dist}(D, D') \leq r$.*

Proof. Let D'_0 be the disk at the leaf w_0. By construction, we have

$$\rho_{D'_0} \leq \rho_{D'}. \tag{3}$$

By construction, D' is the disk in $\mathcal{U}^+_{w_0}$ nearest to D in the inter-center distance. That is,

$$\|c_D - c_{D'}\| \leq \|c_D - c_{D'_0}\|. \tag{4}$$

Assume by contradiction that $\mathsf{dist}(D, D') > r$. Then, by construction,

$$\|c_D - c_{D'_0}\| - \rho_{D'_0} - \rho_D = \mathsf{dist}(D, D'_0) \leq r < \mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_{D'} - \rho_D.$$

That is,

$$\|c_D - c_{D'_0}\| - \rho_{D'_0} < \|c_D - c_{D'}\| - \rho_{D'}. \tag{5}$$

Adding (3) and (5), we get a contradiction to (4). This establishes the lemma. ◀

The tree $T_{\mathcal{R}}$ is defined in an analogous manner for the disks of $\mathcal{R}$, except that (a) the leaves are sorted in increasing order of $\delta(D) + \rho_D$, and (b) the Voronoi diagram $\mathrm{Vor}_2(\mathcal{R}_v)$ at a node v of $T_{\mathcal{R}}$, where $\mathcal{R}_v$ is the set of disks stored at the root of the subtree rooted at v, is the additively-weighted diagram on $\mathcal{R}_v$, where the additive weight of a disk D is $\delta(D)$.

The first kind of Voronoi diagrams $\mathrm{Vor}_1(\mathcal{R}_v)$ are defined exactly as in the case of $T_{\mathcal{U}}$, with the additive weight being $-\rho_D$.

Here too, when we query $T_{\mathcal{R}}$ with a disk D' of $\mathcal{U}$, we search in the tree to find the leftmost leaf w_0 of $T_{\mathcal{R}}$ whose disk D_0 satisfies $\mathsf{dist}(D_0, D') = \|c_{D_0} - c_{D'}\| - \rho_{D_0} - \rho_{D'} \leq r$. This is performed as follows. We first search for the leftmost leaf w_0 whose associated disk D_0 satisfies $\mathsf{dist}(D_0, D') \leq r$, using the same technique as in the previous search in $T_{\mathcal{U}}$, in which we query various Voronoi diagrams $\mathrm{Vor}_1(\mathcal{R}_v)$ along the search path to w_0. We then obtain the set $\mathcal{R}^+_{w_0}$ of all disks stored at the leaves to the right of w_0, including w_0, as the disjoint union of $O(\log n)$ subtrees. We query each of the diagrams $\mathrm{Vor}_2(\mathcal{R}_v)$ associated with these subtrees, and return the disk D that is the nearest neighbor to D' over all these diagrams.

The correctness of this procedure is a consequence of the following "sister" lemma to Lemma 5.

▶ **Lemma 6.** *In the above procedure, the output disk D satisfies* $\mathsf{dist}(D, D') \leq r$.

Proof. Let D_0 be the disk at the leaf w_0. By construction, we have

$$\delta(D_0) + \rho_{D_0} \leq \delta(D) + \rho_D. \tag{6}$$

By construction, D is the disk in $\mathcal{U}^+_{w_0}$ nearest to D' in the inter-center distance with the additive weight δ. That is,

$$\delta(D) + \|c_D - c_{D'}\| \leq \delta(D_0) + \|c_{D_0} - c_{D'}\|. \tag{7}$$

Assume by contradiction that $\mathsf{dist}(D, D') > r$. Then, by construction,

$$\|c_{D_0} - c_{D'}\| - \rho_{D_0} - \rho_{D'} = \mathsf{dist}(D_0, D') \leq r < \mathsf{dist}(D, D') = \|c_D - c_{D'}\| - \rho_D - \rho_{D'}.$$

That is,

$$\|c_{D_0} - c_{D'}\| - \rho_{D_0} < \|c_D - c_{D'}\| - \rho_D. \tag{8}$$

Adding (6) and (8), we get

$$\delta(D_0) + \|c_{D_0} - c_{D'}\| < \delta(D) + \|c_D - c_{D'}\|,$$

which is a contradiction to (7). This establishes the lemma. ◀

We make these nearest-neighbor data structures dynamic by maintaining the various Voronoi diagrams $\mathrm{Vor}_1(\mathcal{U}_v)$, $\mathrm{Vor}_2(\mathcal{U}_v)$, $\mathrm{Vor}_1(\mathcal{R}_v)$, $\mathrm{Vor}_2(\mathcal{R}_v)$, dynamically, using the technique of [8, 11], in which the update time of each diagram is $O(\log^4 n)$. An update of either of the trees $T_{\mathcal{U}}$, $T_{\mathcal{R}}$ requires updating $O(\log n)$ data structures along the path to the leaf containing the inserted or deleted element, and therefore takes $O(\log^5 n)$ time. The transformation of Chan [6] from the two nearest-neighbor structures to a BCP data structure incurs an additional logarithmic overhead. Overall we get

▶ **Theorem 7.** *The shortest-path tree from some starting disk in the proximity graph of n disks (of arbitrary radii) in the plane, under the weights of inter-center distances, can be constructed in $O(n \log^6 n)$ time.*

Remark. As in the unweighted version, in the case of unit disks, the decision procedure itself can be implemented to run faster than $O(n \log^6 n)$. For example, Wang and Xue [13] present an algorithm that runs in $O(n \log^2 n)$ time, but this algorithm is not suitable for our optimization procedure, since its comparisons generate critical values that are determined by more than two disks. We thus replace its "problematic" components by new ones, to obtain a (somewhat simpler) algorithm that is suitable for the optimization procedure, see the full version of this paper.

4.2 The optimization procedure

The most natural reverse shortest path question is to find the smallest value of r such that the length of the shortest path in G_r from D_s to D_t is smaller than some given real parameter w.

This optimization procedure is implemented as in the unweighted case, using the same combination of the interval shrinking and bifurcation procedures. The critical values are the same as in the case for the unweighted proximity graph.

In summary, we have shown:

▶ **Theorem 8.** *The reverse shortest path problem for weighted intersection or proximity graphs of arbitrary disks in the plane can be solved in $O^*(n^{5/4})$ randomized expected time. This bound applies to all the variants of the problem listed above. In the case of unit disks, where all radii are equal, the problem can be solved in $O^*(n^{6/5})$ randomized expected time.*

Remark. The machinery developed in this section seems to be more broadly applicable to other optimization questions that involve shortest paths in weighted proximity or intersection graphs, of disks or of more general geometric objects. Simple extensions could involve different definitions of proximity or the use of other weight functions. Other extensions could involve different problems, such as computing all-pairs shortest paths, computing shortest paths with negative edge weights, computing the diameter of the graph, etc.

References

1 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

2 P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. *Algorithmica*, 9:495–514, 1993.

3 P. K. Agarwal, M. J. Katz, and M. Sharir. On reverse shortest paths in geometric proximity graphs. In *33rd Int. Sympos. on Algorithms and Computation (ISAAC)*, pages 42:1–42:19, 2022.

4 R. Ben Avraham, O. Filtser, H. Kaplan, M. J. Katz, and M. Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Trans. Algorithms*, 11, 2015, Art. 29.

5 S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom. Theory Appls.*, 48:360–367, 2015.

6 T. M. Chan. Dynamic generalized closest pair: Revisiting Eppstein's technique. In *Proc. 3rd Sympos. Simplicity in Algorithms (SOSA)*, pages 33–37, 2020.

7 T. M. Chan and D. Skrepetos. All-pairs shortest paths in unit disk graphs in slightly subquadratic time. In *Proc. 27th Int. Sympos on Algorithms and Computation (ISAAC)*, pages 24:1–24:13, 2016.

8 H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Dynamic planar voronoi diagrams for general distance functions and their algorithmic applications. *Discrete Comput. Geom.*, 64:838–904, 2020.

9 M. J. Katz and M. Sharir. Efficient algorithms for optimization problems involving semi-algebraic range searching. in `arXiv:2111.02052`.

10 M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26:1384–1408, 1997.

11 C.-H. Liu. Nearly optimal planar k nearest neighbors queries under general distance functions. *SIAM J. Comput.*, 51(3):723–765, 2022.

12 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. of the ACM*, 30:852–865, 1983.

13 H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete Comput. Geom.*, 64:1141–1166, 2020.

14 H. Wang and Y. Zhao. Reverse shortest path problem for unit-disk graphs. In *Proc. 17th Algorithms and Data Structures Sympos. (WADS)*, pages 655–668, 2021.

15 H. Wang and Y. Zhao. Reverse shortest path problem in weighted unit-disk graphs. In *Proc. 16th Int. Conf. and Workshops on Algorithms and Computation (WALCOM)*, pages 135–146, 2022.