BFS and Reverse Shortest Paths for Ball Intersection Graphs in Three and Higher Dimensions

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 and AI, Tel Aviv University, Israel

Abstract

Let $\mathcal{B}$ be a collection of n arbitrary balls in $\mathbb{R}^3$, and let $G_0(\mathcal{B})$ be their intersection graph. We provide an algorithm for performing BFS on $G_0(\mathcal{B})$, which runs in $O^*(n^{4/3})$ time, where the $O^*(\cdot)$ notation hides subpolynomial factors. For $r \geq 0$, let $G_r(\mathcal{B})$ be the intersection graph of the set $\mathcal{B}_r = \{B + r \mid B \in \mathcal{B}\}$, where $B + r$ is the ball concentric with B whose radius is larger by r than the radius of B. We provide an efficient algorithm for the reverse shortest path (RSP) problem, where we are given two designated balls B_s, B_t of $\mathcal{B}$ and a parameter $0 < \lambda < n$, and seek the smallest value r^* for which $G_{r^*}(\mathcal{B})$ contains a path from B_s to B_t of at most λ edges. For the special case of congruent balls (equivalently, for points in $\mathbb{R}^3$), the algorithm runs in $O^*(n^{29/21}) \approx O^*(n^{1.381})$ time. For the general case, the algorithm runs in $O^*(n^{56/39}) \approx O^*(n^{1.436})$ time. We also extend the technique to handle other measures of expansion and higher dimensions.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, reverse shortest paths, breadth-first search, shrink-and-bifurcate, intersection graphs

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.45

Funding *Matthew J. Katz*: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation, and by Grant 495/23 from the Israel Science Foundation.
Rachel Saban: Work partially supported by Grant 495/23 from the Israel Science Foundation.
Micha Sharir: Work partially supported by Grant 495/23 from the Israel Science Foundation.

1 Introduction

The reverse shortest path (RSP) problem has received considerable attention recently. In general, we are given a set P of n geometric objects, and a parameter $r \geq 0$. We define a graph $G_r(P)$ on P, whose edges are all the pairs (u, v) that satisfy some property, given by a predicate $\Pi(u, v; r)$, which is monotone in r, meaning that if $\Pi(u, v; r)$ holds then $\Pi(u, v; r')$ holds for all $r' > r$. That is, the graphs $G_r(P)$ are monotone increasing in r. We are given two designated objects $s, t \in P$ and a parameter $0 < \lambda < n$, and wish to find the smallest value r^* for which $G_{r^*}(P)$ contains a path from s to t with at most λ edges.

Among the simplest examples is the case where P is a set of n points in the plane, and $G_r(P) = \{(u, v) \mid |uv| \leq r\}$, under the Euclidean distance $|uv|$. This can also be interpreted as the intersection graph of the disks of radius $r/2$ centered at the points of P (the so-called unit-disk graph, with the unit being $r/2$). This case and many variants thereof are reviewed later in the introduction.

One generalization, studied here, is to the case of points in three dimensions. That is, $G_r(P)$ in this case is the intersection graph of n congruent balls of radius $r/2$ centered at the points of P. This is a special instance of the general problem studied in this work, where the balls have arbitrary radii. That is, the input consists of a set $\mathcal{B}$ of n balls in $\mathbb{R}^3$ of arbitrary radii, and r is a parameter that measures how the balls of $\mathcal{B}$ are expanded. The simplest case is where we increase the radius of each ball by adding r to it. Let $G_r(\mathcal{B})$ denote the intersection graph of the expanded balls, so $G_0(\mathcal{B})$ is the intersection graph of the original balls. Our technique also applies to any other well-behaved measure of expansion, e.g., multiplying each radius by $r \geq 1$, but for conciseness we will mainly stick to the additive measure of expansion, as just introduced, and discuss the general setup later in the paper.

A standard high-level approach to the RSP problem is to design a decision procedure which, with r as input, determines whether $G_r(\mathcal{B})$ contains a path from s to t of length at most λ. If it does then $r^* \leq r$, and if not then $r^* > r$. The decision procedure is typically implemented using Breadth-First Search (BFS) on $G_r(\mathcal{B})$. Once we have such a procedure, we can use binary search on r to home in on the correct value of r^*.

There are two major problems with this approach. First, the cost of a naïve implementation of BFS is linear in the number of edges of $G_r(\mathcal{B})$, which can be $\Theta(n^2)$ in the worst case. Second, BFS appears not to be parallelizable, and thus it is not amenable to parametric search, which is the standard technique for turning the decision procedure into an efficient optimization procedure for solving the RSP problem. In this work we overcome these issues, for the special case of ball-intersection graphs in $\mathbb{R}^3$ (and in higher dimensions), and obtain efficient algorithms for solving the problem, albeit they are (not surprisingly) less efficient than in the planar case.

Related work. Breadth-First Search (BFS) is recognized as one of the fundamental graph algorithms. On general graphs $G = (V, E)$, its running time is $O(|V|+|E|)$, which is inefficient when $|E|$ is large. Its importance has led authors to seek families of graphs in which BFS can be implemented more efficiently. In the geometric context, one such example is the family of disk graphs in the plane. The vertices of a disk graph represent disks in the plane and there exists an edge between two vertices if and only if the corresponding disks intersect. Notice that the number of edges in a disk graph can be quadratic in n, the number of disks. Nevertheless, for unit-disk graphs (where all the disks are congruent), Cabello and Jejčič [8] presented an $O(n \log n)$ implementation of BFS, and subsequently Chan and Skrepetos [10] presented an alternative $O(n)$ implementation (after pre-sorting the points by their x- and y-coordinates). For arbitrary disks, Klost [14] described an $O(n \log^2 n)$ implementation of BFS (see also [11, 12]).

The proximity graph of a set S of n segments in the plane, for a real parameter $r \geq 0$, is $G_r(S) := (S, E)$, where $E = \{(e_1, e_2) \mid \text{dist}(e_1, e_2) \leq r\}$ and $\text{dist}(e_1, e_2)$ is the Euclidean distance between e_1 and e_2 (which is 0 if they intersect). Agarwal et al. [5] devised an $O^*(n^{4/3})$ implementation of BFS in $G_r(S)$. (As in the abstract, the $O^*(\cdot)$ notation hides subpolynomial factors.) For the special case where the segments in S are pairwise disjoint, Agarwal et al. [4] have later provided an $O(n \log^2 n)$ implementation.

The bottleneck path problem (also known as the minimax path problem) and its complementary problem, i.e., the widest path problem (or the maximum capacity problem), are well-known problems in graph theory. In suitable contexts, the bottleneck path problem is strongly related to the RSP problem: Given s and t, if π is a path from s to t of at most λ edges with the minimum bottleneck r^*, then r^* is the solution to the RSP problem with s, t, λ as input, and vice versa.

Abu-Affash et al. [1] studied the problem of placing at most k Steiner points to minimize the bottleneck of a path between two designated points in the plane, for a given integer parameter $k \geq 0$, and developed an $O(n \log^2 n)$-time algorithm for the problem.

The RSP problem for unit disks (as mentioned above) was studied by Wang and Zhao [16], who proposed an $O^*(n^{5/4})$-time solution. An improved solution with running time $O^*(n^{6/5})$, using the shrink-and-bifurcate technique (briefly reviewed later; see [7]), was subsequently presented by Kaplan et al. [11]. A very recent improvement of this technique by Chan and Huang [9] yields an implementation with running time $O^*(n^{8/7})$, which can be further improved, for the case of unit-disk graphs, to $O^*(n^{9/8})$ time. For arbitrary disks, Kaplan et al. [11] obtained a solution with randomized expected time $O^*(n^{5/4})$, which can be improved to randomized expected time $O^*(n^{6/5})$, using Chan and Huang's technique [9].

The RSP problem was also studied for other objects, including wedges of some fixed angle, which are viewed as directional antennas, and unit-height towers placed on a 1.5-dimensional terrain. The former version was solved in $O^*(n^{4/3})$ time by Agarwal et al. [5], and the latter version was studied in Katz et al. [13] (see also [9]).

Our results. Our main contributions are efficient algorithms for the BFS and RSP problems in ball-intersection graphs in three and higher dimensions. We first consider the problem of efficient implementation of BFS, as a decision procedure for guiding the search for the optimum r^* in the RSP problem (and also an interesting problem in itself). In three dimensions we can perform BFS on $G_0(\mathcal{B})$, for a set $\mathcal{B}$ of n (congruent or arbitrary) balls in $\mathbb{R}^3$, in $O^*(n^{4/3})$ time, and in d dimensions in $O^*(n^{2e/(e+1)})$ time, where $e = \lfloor d/2 \rfloor + 1$. As these algorithms are not parallelizable (at least we do not know how to run them in small parallel depth),[1] we need to use a different approach. Such an approach, already mentioned above, known as the *shrink-and-bifurcate* technique, has originally been developed in Ben Avraham et al. [7] for computing a rather special case of Fréchet distances. But it has recently found applications to the RSP problem in the plane [11]. Very recently, as already mentioned, it has been dramatically improved by Chan and Huang [9].

Once a BFS procedure (serving as the decision procedure) is available, our technique follows closely the improved technique in [9], which however requires certain nontrivial modifications and enhancements to fit into higher-dimensional contexts. The technique of [9] consists of two parts, a general-purpose part, and an additional enhancement which improves the running time further for the special case of unit-disk graphs. We follow the general-purpose part for arbitrary balls, and both parts for congruent balls.

In three dimensions, for the special case of congruent balls (equivalently, of a set of n points in $\mathbb{R}^3$), we obtain an algorithm that runs in $O^*(n^{29/21})$ randomized expected time. For the case of balls of arbitrary radii, we obtain an algorithm that runs in $O^*(n^{56/39})$ randomized expected time. More generally, in $d \geq 3$ dimensions, the algorithm for arbitrary balls runs in

$$O^*\left(n^{\frac{2(d+1)(3e+1)}{(3d+4)(e+1)}}\right)$$

randomized expected time, and, for the special case of congruent balls, the algorithm runs in randomized expected time

$$O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right), \text{ where } g = \frac{2(d-e)}{(e+1)(3d+1)}.$$

[1] This is a standard issue in BFS on any graph.

2 BFS in ball-intersection graphs

Let $\mathcal{B}$ be a collection of n balls in $\mathbb{R}^3$, of arbitrary radii.[2] We denote a ball with center c and radius ρ as $B(c,\rho)$. Let $G_0(\mathcal{B})$ be the intersection graph of $\mathcal{B}$, i.e., the vertices of $G_0(\mathcal{B})$ are the balls of $\mathcal{B}$, and its edges consist of all the intersecting pairs of balls. The condition for two balls $B(c_1,\rho_1)$, $B(c_2,\rho_2)$ to intersect is $|c_1c_2| \leq \rho_1+\rho_2$.

Let B_s be a start ball in $\mathcal{B}$. Our goal is to run BFS on $G_0(\mathcal{B})$ starting from B_s. We follow the standard approach, of constructing the layers of the BFS in order, starting from layer L_0 that contains only B_s. Consider the step of passing from some layer L_i to the next layer L_{i+1}. Let $\mathcal{U}_i$ denote the set of all balls that the BFS has not yet reached up to this step. We then need to find, for each ball $B=B(c,\rho)$ in L_i, all the balls of $\mathcal{U}_i$ that it intersects. Each such ball is added to L_{i+1} and is immediately deleted from $\mathcal{U}_i$, to ensure that it is not detected again by other balls of L_i (or by balls in future layers). We continue in this way until no more balls of $\mathcal{U}_i$ intersect any ball in L_i. At this point L_{i+1} has been computed, and we go on to construct the next layer. (For our decision procedure, executed on $G_r(\mathcal{B})$, we stop when either the target ball B_t has been reached or layer L_λ has been constructed, whichever happens first. If B_t has been reached, we report success (i.e., $r^* \leq r$), otherwise we report failure (i.e., $r^* > r$).)

To implement this algorithm efficiently, we therefore need a dynamic data structure for storing $\mathcal{U}=\mathcal{U}_i$, the set of unreached balls, that supports (efficiently) the following two kinds of operations.

Intersection detection queries: Given a query ball B, detect whether it intersects any ball of $\mathcal{U}$, and, if so, report such a ball.

Deletions: Delete a ball B from $\mathcal{U}$.

An intersection query with a ball $B(c_0,\rho_0)$ can be rephrased as a range emptiness detection query. For this, we map each ball $B=B(c,\rho)=B((x_c,y_c,z_c),\rho)$ of $\mathcal{U}$ to the point $\hat{B}=(x_c,y_c,z_c,\rho)$ in $\mathbb{R}^4$, and map each ball $B_0=B(c_0,\rho_0)=B((x_{c_0},y_{c_0},z_{c_0}),\rho_0)$ of L_i to the range

$$\sigma_{B_0}=\{(x,y,z,\rho)\in\mathbb{R}^4 \mid (x-x_{c_0})^2+(y-y_{c_0})^2+(z-z_{c_0})^2 \leq (\rho+\rho_0)^2\}.$$

Then a ball B intersects B_0 iff $\hat{B}\in\sigma_{B_0}$.

Using a standard lifting transform, we further map each point $\hat{B}=(x_c,y_c,z_c,\rho)$ to the point $B^*=(x_c,y_c,z_c,\rho,x_c^2+y_c^2+z_c^2-\rho^2)$ in $\mathbb{R}^5$. Each range σ_{B_0} is lifted to the halfspace

$$h_{B_0} := \quad x_5 \leq 2x_{c_0}x+2y_{c_0}y+2z_{c_0}z+2\rho_0\rho+(\rho_0^2-x_{c_0}^2-y_{c_0}^2-z_{c_0}^2).$$

As is easily checked, $\hat{B}\in\sigma_{B_0}$ iff B^* lies in h_{B_0}.

In other words, in the transformed problem we have a set $\mathcal{B}^*=\{B^* \mid B\in\mathcal{B}\}$ of n points in $\mathbb{R}^5$, which we want to process into a dynamic data structure (under deletions) for answering halfspace range emptiness queries. The algorithm of Agarwal and Matoušek [6] provides such a structure. Specifically, for any given parameter $n\leq s\leq n^2$, the structure can be constructed to have size $O^*(s)$, an initial version of it can be constructed in $O^*(s)$ time, each query takes $O^*(n/s^{1/2})$ time, and each deletion takes amortized $O^*(s/n)$ time. Choosing $s=n^{4/3}$, we obtain a data structure of size $O^*(n^{4/3})$, so that each operation (query or deletion) on the structure takes $O^*(n^{1/3})$ time.

[2] The case of congruent balls does not lead to an improved BFS implementation, although the algorithm becomes considerably simpler; see a discussion at the end of the section.

The number of operations is $O(n)$, since each query either detects a new intersecting ball, which is promptly removed from $\mathcal{U}$, or is the last query performed with a ball, and each ball becomes a query only at the layer it belongs to. We conclude that the overall running time of the BFS is $O^*(n^{4/3})$, because the cost of maintaining and searching in the data structure dominates the cost of the other steps of the BFS. We thus obtain:

▶ **Theorem 1.** *BFS on the ball intersection graph of a set of n balls in $\mathbb{R}^3$ can be performed in $O^*(n^{4/3})$ time.*

The case of congruent balls. We first state the following theorem, whose results will be useful for handling congruent balls. Its proof is an easy consequence of the analysis in [6, 15], applied to the lifted points and halfspaces in $\mathbb{R}^5$, as above.

▶ **Theorem 2.** *(a) Given two sets P, Q of m and n points, respectively, in three dimensions, and a prameter $r > 0$, we can determine, for each $p \in P$ whether there exists a point $q \in Q$ such that $|pq| \leq r$, in overall time*

$$O^*\left(m^{2/3}n^{2/3} + m + n\right).$$

(b) Given two sets P, Q, as above, we can compute, for each $p \in P$, the nearest neighbor of p in Q, in overall time

$$O^*\left(m^{2/3}n^{2/3} + m + n\right).$$

(Note that (b) subsumes (a), but the algorithm in (a) is simpler, so we state it separately.)

When the balls of $\mathcal{B}$ are congruent, say all of radius r, we do not need the above dynamic (and rather involved) structure. Instead, we follow the approach in the planar setup of unit-disk graphs [8, 10]. That is, we form a grid of cell size $r/2$, and distribute the ball centers among its cells. When we perform the step of passing from L_i to L_{i+1}, we process each *active* grid cell, i.e., a cell that contains centers of balls in L_i. For each such cell τ, all unreached balls with centers in τ are immediately placed in L_{i+1}. Then, for each cell τ' in a suitable punctured constant-size neighborhood of τ (only such cells, and τ itself, can contain centers of balls in $\mathcal{U}_i$ that can be reached, in one step, from balls with centers in τ), we take the set $\mathcal{U}_i(\tau')$ of unreached balls with centers in τ' and test each such ball B whether its center lies within distance r from the center of some ball in $L_i(\tau)$, the set of balls of L_i with centers in τ. Theorem 2(a) implies that this reverse search can be implemented in time

$$O^*\left(|\mathcal{U}_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |\mathcal{U}_i(\tau')| + |L_i(\tau)|\right).$$

The remaining straightforward implementation details, and the analysis of correctness and performance of the structure, as given in [8, 10, 11], carry over to the three-dimensional case, and imply that the overall cost of this implementation of the BFS is $O^*(n^{4/3})$ time.

3 Reverse shortest paths in unit-ball graphs in three dimensions

We can now solve the corresponding optimization problem, which is the *reverse shortest path (RSP) problem*, or, as it is also called, the *bounded-hop bottleneck path problem*, for unit-ball graphs in $\mathbb{R}^3$: Given a set P of n points in $\mathbb{R}^3$, two designated points $s, t \in P$, and a parameter $0 < \lambda < n$, find the minimum value r^* of r for which the associated graph $G_{r^*}(P)$ contains a path from s to t of at most λ edges.

We can solve the RSP problem using the aforementioned shrink-and-bifurcate technique [7, 9, 11]. We begin with a brief overview of the technique. It applies in more general setups, but let us restrict it to our setup of unit-ball graphs (and later also to intersection graphs of arbitrary balls, although the description below assumes that the balls are congruent). The shrinking stage constructs an interval I that contains r^* and at most L other critical values of r, each of which is the distance between two centers, where L is a prespecified parameter. It does so, in the improved approach of [9], by running a distance selection algorithm on various pairs of random samples of $\mathcal{B}$ (that is, on the sets of centers of the balls in the samples).

The distance selection algorithm, on which the shrinking procedure is based, takes, ignoring the shrinking aspect, $O^*(n^{3/2})$ time. More generally, it takes $O^*(m^{3/4}n^{3/4} + m + n)$ time for selecting distances between two sets of m and n points, respectively, which is the setup that arises when applying the technique of [9]. Using (a suitable adaptation to three dimensions of) the shrinking mechanism of Chan and Huang [9], the construction of the desired interval I can be performed in expected $O^*(n^{3/2}/L^{3/4})$ time.

This is followed by a bifurcation stage, in which we simulate the decision procedure (in our setup, the BFS procedure of the preceding section). When we reach a comparison, we resolve it right away if the critical value r that it induces lies outside I. If however r lies in I, we bifurcate, exploring both outcomes $r^* \le r$ and $r^* > r$. When we have either accumulated enough unresolved comparisons, or moved forward at least some number of steps in the simulation, along each path in the bifurcation tree, we stop and resolve all comparisons using binary search, guided by the (unsimulated) decision procedure. This shrinks I further, and we start a new phase of bifurcating simulation. As shown in [7, 11], with a suitable choice of parameters, the overall cost of the bifurcation is $O^*(L^{1/2}D(n))$, where $D(n) = O^*(n^{4/3})$ is the cost of the decision procedure (i.e., of the BFS).

Hence the overall cost of the RSP algorithm is $O^*(n^{3/2}/L^{3/4} + L^{1/2}n^{4/3})$. We choose L to balance these terms, that is, $L = n^{2/15}$, and then the running time is $O^*(n^{7/5})$. That is, we have our initial, albeit weaker, result:

▶ **Theorem 3.** *For a set P of n points in $\mathbb{R}^3$, two designated points $s, t \in P$, and an integer parameter $0 < \lambda < n$, we can find the minimum radius r^* for which the associated graph $G_{r^*}(P)$ contains a path from s to t of at most λ edges, in $O^*(n^{7/5})$ time.*

3.1 An improved solution

To obtain a faster algorithm, we follow the high-level approach of the second improvement in [9], with many nontrivial adjustments that cater to the three-dimensional nature of the problem. We follow the same setup as in the second implementation of the BFS algorithm. However, for technical reasons that will become clear later on, we first perform the following steps, having to do with the cell size of the grid. We first note that the precise size of the cells (which we have set to $r/2$ in the BFS algorithm) is not very important, as long as it is smaller than $r/\sqrt{3}$ (but not much smaller), so that the propagation within a cell is immediate. We can therefore proceed as follows. Note that, by the nature of the RSP problem, r^* must be between $|st|/2$ and $|st|/(2n)$ (or, more precisely, $|st|/(2\lambda)$). We therefore run binary search through this interval, using any of the BFS decision procedures of Section 2 to guide the search, and in $O(\log n)$ steps we locate r^* within a range $[r_1, (1+\varepsilon)r_1]$, for some suitable

small $\varepsilon > 0$. We then set r to be[3] $r_1/2$, and keep this size throughout the simulation. See below how this is used in the algorithm. We emphasize that this step precedes the actual simulation, as well as the step of distributing the points of P in the grid cells.

We introduce a new threshold parameter $\Delta \ll n$, whose concrete value will be set later, and classify each nonempty grid cell as either *heavy*, if it contains at least Δ points of P, or *light*, otherwise. The number of heavy cells is at most $k := n/\Delta$. We now compress each heavy cell τ into a single (symbolic) "heavy" point p_τ, and obtain a new compressed version of $G_r(P)$, which we denote as $H_r(\hat{P})$, where $\hat{P}$ consists of all the points in the light cells and of all the representative heavy points p_τ. The distance between any pair p, q of points in light cells remains the same, namely $|pq|$. The distance between a heavy point p_τ and a light point q (a point in a light cell) is $\mathrm{dist}(q, P_\tau) = \min_{p \in \tau} |qp|$. Finally, the distance between two heavy points p_τ, $p_{\tau'}$ is the distance determined by the bichromatic closest pair (BCP) in $P_\tau \times P_{\tau'}$.

Running BFS on $H_r(\hat{P})$ is performed similarly to the previous implementation. A single propagation step, at some iteration i, involving two adjacent cells τ, τ', is executed depending on whether one of τ, τ', or both, are heavy. If both cells are light, we proceed as before, incurring the cost

$$O^*\left(|U_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |U_i(\tau')| + |L_i(\tau)|\right); \tag{1}$$

see Theorem 2(a). When τ is light and τ' is heavy, $U_i(\tau')$ (if nonempty) is the singleton representative point $p_{\tau'}$, and by running the all-nearest-neighbor procedure (Theorem 2(b)) on $L_i(\tau)$ and $P_{\tau'}$, we can determine whether this step should place $p_{\tau'}$ in L_{i+1}. The cost is then

$$O^*\left(|P_{\tau'}|^{2/3}|L_i(\tau)|^{2/3} + |P_{\tau'}| + |L_i(\tau)|\right). \tag{2}$$

A similar procedure is applied when τ is heavy and τ' is light. In this case each point of $U_i(\tau')$ searches for a near neighbor in P_τ, and the cost is

$$O^*\left(|U_i(\tau')|^{2/3}|P_\tau|^{2/3} + |U_i(\tau')| + |P_\tau|\right). \tag{3}$$

Finally, when both cells are heavy we need to compute the distance of the BCP in $P_\tau \times P_{\tau'}$. This step is handled differently because its naïve implementation is too expensive, because it does not exploit the smaller parameter Δ; see shortly below.

We sum the bounds in (1), (2) and (3), over all iterations i and pairs τ, τ' of adjacent cells that are active at the corresponding iteration, when both τ and τ' are light (eq. (1)), when τ is light and τ' is heavy (3), and when τ is heavy and τ' is light (2). Note that in each of the bounds (1)–(3), at least one of the sets ($L_i(\tau)$ and / or $U_i(\tau')$) is from a light cell, so its size is at most Δ.

Applying Hölder's inequality, one can show that the sums of the bounds in (1), (2) and (3) are all $O^*(n\Delta^{1/3})$. For example, summing the bound in (1) over all iterations i and active pairs τ, τ' of neighboring cells, and using the facts that (i) $|L_i(\tau)| \leq \Delta$, and (ii) each cell is active, or a neighbor of an active cell, at only $O(1)$ iterations, in each of these bounds, we get

[3] Note that this choice might force us to change the definition of the neighborhood of a cell to potentially contain more cells, but still only a constant number thereof.

$$O^*\left(\sum_i \sum_{\tau,\tau'} \left(|U_i(\tau')|^{2/3}|L_i(\tau)|^{2/3} + |U_i(\tau')| + |L_i(\tau)|\right)\right)$$
$$= O^*\left(\Delta^{1/3} \sum_i \sum_{\tau,\tau'} |U_i(\tau')|^{2/3}|L_i(\tau)|^{1/3}\right) + O^*(n)$$
$$= O^*\left(\Delta^{1/3} \left(\sum_i \sum_{\tau,\tau'} |U_i(\tau')|\right)^{2/3} \left(\sum_i \sum_{\tau,\tau'} |L_i(\tau)|\right)^{1/3}\right) + O^*(n) = O^*(n\Delta^{1/3}),$$

and similarly for the sums of the bounds in (2) and (3).

It remains to handle the case where both τ and τ' are heavy. Here we need to compute the distance of the BCP in $P_\tau \times P_{\tau'}$, over all adjacent heavy pairs (i.e., pairs where both τ and τ' are heavy). As above, this can be done in a total of $O^*(n^{4/3})$ time. Since it does not depend on Δ, this cost turns out to be too expensive to pay during the simulation of the procedure. To bypass this issue, we note that, once we have fixed the cell-size $r_1/2$ of the grid (as we did, ahead of the simulation), this step is independent of r or r^*, and can too be performed ahead of the simulation. We thus compute the BCPs for all pairs of adjacent heavy cells, once and for all. The overall cost, of both the binary search to fix the cell-size of the grid and the BCP computations, is easily seen to be $O^*(n^{4/3})$. In addition, after having computed all the heavy BCP distances, we run an additional binary search through them, to ensure that the interval with which we start the shrink-and-bifurcate procedure does not contain any heavy BCP distance in its interior.

All these considerations imply that we can run BFS on the compressed graph $H_r(\hat{P})$ in time $O^*(n\Delta^{1/3})$, after an initial phase that takes $O^*(n^{4/3})$ time, which we will execute only once, outside (and preceding) the actual simulation during the search for r^*.

We now run this simulation of the BFS on $H_{r^*}(\hat{P})$, using the shrink-and-bifurcate paradigm, as in [9] and as briefly reviewed above. As before, the shrinking procedure is based on distance selection in $\mathbb{R}^3$, but we apply it separately to each pair of adjacent grid cells, at least one of which is light. (This is justified as in [9], because r^* must arise as a distance within such a pair, unless it is the BCP distance of some heavy pair, which will have been detected during the preliminary stage.) Arguing as in [9], and adapting the setup to $\mathbb{R}^3$, the cost of the shrinking stage is $O^*(n\Delta^{1/2}/L^{3/4})$.

The subsequent bifurcation procedure runs, as before, in time $O^*(L^{1/2}D(n))$, where $D(n)$ is the cost of the decision procedure. Ignoring the cost of the preliminary part, which stays outside the simulation, we have $D(n) = O^*(n\Delta^{1/3})$. Hence the overall cost of the shrink-and-bifurcate procedure, adding back the preliminary cost as a one-time cost, is

$$O^*(n\Delta^{1/2}/L^{3/4} + L^{1/2}n\Delta^{1/3} + n^{4/3}). \tag{4}$$

Of course, so far we have only simulated the BFS on the compressed graph $H_{r^*}(\hat{P})$, so the output value r^* is still not the correct value. Denoting it by r_1^*, it is the smallest value of r for which $H_r(\hat{P})$ has a path of at most λ edges between s and t. As argued in [9], denoting by $d_H(s,p)$ the graph distance between s and a point p in $H_{r_1^*}(\hat{P})$, and by $d_G(s,p)$ the graph distance between s and p in $G_{r^*}(P)$, we have, for each p,

$$d_G(s,p) - k \le d_H(s,p) \le d_G(s,p), \tag{5}$$

where $k = n/\Delta$. With some care. this also covers the cases where s and / or t lie in heavy cells.

Recovering r^*

As in [9], to find the true value r^*, we use dynamic programming, following the high-level approach of [9]. For the sake of completeness, we repeat here some details of this technique, adapted to our setup. We first comment that it is easy to solve the RSP problem using dynamic programming, without using the preceding machinery at all. To do so, let $D_i(p)$, for $p \in P$, denote the smallest bottleneck value of a path from s to p consisting of at most i edges. Initially, $D_0(s) = 0$ and $D_0(p) = +\infty$ for any $p \neq s$. Then we have, for $i \geq 1$ and $p \in P$,

$$D_i(p) = \min_{q \in P} \max \{D_{i-1}(q),\ |qp|\}\,. \tag{6}$$

If $D_\lambda(t) < +\infty$ then $D_\lambda(t) = r^*$, which follows easily from (6). The problem with this approach is that the number of times the step (6) is applied is $\Theta(n^2)$ (it is actually $O(n\lambda)$, for the number of points n and the number of iterations λ to reach, or not to reach, t). Moreover, without any efficient implementation of the steps (6), the overall cost can be $\Theta(n^3)$.

In our setup, though, we can reduce the number of dynamic programming steps to $O(nk)$. Indeed, write $P_i = \{p \in P \mid i - k \leq d_H(p) \leq i\}$. As each point participates in $k+1$ sets P_i, we have $\sum_i |P_i| = O(nk)$.

As in [9], we exploit this property by replacing the function D by another function $\hat{D}$, defined by the following modified iterative process. For each $p \in P_i$ put

$$\hat{D}_i(p) = \min_{q \in P_{i-1}} \max \left\{\hat{D}_{i-1}(q),\ |qp|\right\}, \tag{7}$$

and $\hat{D}_0(s) = 0$ and $\hat{D}_0(p) = +\infty$ for all $p \neq s$. As shown in [9], we also have $\hat{D}_\lambda(t) = r^*$. (This argument is fully general and discrete, and does not depend on any geometric feature of the problem, except for the inequalities (5). In particular, it does not depend on the dimension.)

The number of steps in this modified dynamic programming process is thus $O(nk)$. To make these steps (relatively) efficient, we follow a variant of the technique in [9, Lemma 8], adapted to three dimensions. Briefly, the input is a set Q of n points in $\mathbb{R}^3$, so that each point $q \in Q$ carries a positive weight w_q (the weight w_q will be set to $\hat{D}_{i-1}(q)$ at the i-th iteration), and we have a set P of m queries, where each query point p seeks the point $q \in Q$ that minimizes $\max\{w_q, |qp|\}$. The algorithm in [9, Lemma 8] is based on Voronoi diagrams, which works well in the plane, but is not efficient in three dimensions. We use the following modified approach. As in [9], we store the points of Q at the leaves of a balanced binary tree T, sorted in increasing order of their weights. Each node v of T processes the set Q_v of those points of Q that are stored at the subtree rooted at v. Each query point p follows a path in T. When p reaches a node v, with a left child v_L and a right child v_R, it finds its (standard) nearest neighbor q_L in Q_{v_L}. If $|pq_L| \leq w_{q_L}$ we recurse in v_L, and if $|pq_L| > w_{q_L}$ we recurse in v_R. See [9] for details and justification of this rule.

Instead of using Voronoi diagrams, as in [9], we use the algorithm in Theorem 2(b). To do so, we run all the queries in parallel, proceeding level by level. When we reach a node v, we have available the set P_v of all the queries whose paths through T reach v. Using the algorithm in Theorem 2(b), we find, for each $p \in P_v$, its nearest neighbor in Q_{v_L}, in overall time

$$O^*\left(|P_v|^{2/3}|Q_{v_L}|^{2/3} + |P_v| + |Q_{v_L}|\right).$$

This allows us, by our search rule, to assign each $p \in P_v$ either to v_L or to v_R, as appropriate. Carrying out this process at each node of the current level, we obtain, for each node w of the next level, the set P_w of query points that want to access w (actually, w_L). We continue in this recursive manner until we reach the leaves. The cost at a leaf v is $O(1 + |P_v|)$. Summing over all nodes of T, bearing in mind that

$$\sum_v |P_v| = O(m \log n) \qquad \text{and} \qquad \sum_v |Q_{v_L}| = O(n \log n),$$

the overall cost is easily seen to be $O^*\left(|P|^{2/3}|Q|^{2/3} + |P| + |Q|\right)$.

We now apply this procedure at each iteration i of the dynamic programming, where the input set is P_{i-1}, the query set is P_i, and the weight of each input point q is $\hat{D}_{i-1}(q)$. Clearly, the output of this procedure, with this input, implements the iterative process in (7). The cost of iteration i is therefore

$$O^*\left(|P_i|^{2/3}|P_{i-1}|^{2/3} + |P_i| + |P_{i-1}|\right).$$

Since each $|P_i|$ is at most n, and $\sum_i |P_i| = O(nk)$, the overall cost of the dynamic program is (using Hölder's inequality)

$$\begin{aligned}
&O^*\left(\sum_i \left(|P_i|^{2/3}|P_{i-1}|^{2/3} + |P_i| + |P_{i-1}|\right)\right) \\
&= n^{1/3} O^*\left(\sum_i \left(|P_i|^{1/3}|P_{i-1}|^{2/3}\right)\right) + O^*\left(\sum_i (|P_i| + |P_{i-1}|)\right) \\
&= O^*\left(n^{1/3}(nk)^{1/3}(nk)^{2/3} + nk\right) = O^*(n^{4/3}k) = O^*\left(\frac{n^{7/3}}{\Delta}\right). \qquad (8)
\end{aligned}$$

Altogether, the cost of the full algorithm is therefore

$$O^*\left(\frac{n\Delta^{1/2}}{L^{3/4}} + L^{1/2}n\Delta^{1/3} + n^{4/3} + \frac{n^{7/3}}{\Delta}\right) = O^*\left(\frac{n\Delta^{1/2}}{L^{3/4}} + L^{1/2}n\Delta^{1/3} + \frac{n^{7/3}}{\Delta}\right). \qquad (9)$$

We first balance the first two terms, choosing $L^{5/4} = \Delta^{1/6}$, or $L = \Delta^{2/15}$, and then the bound becomes

$$O^*\left(n\Delta^{2/5} + \frac{n^{7/3}}{\Delta}\right).$$

We now choose Δ to balance these terms, that is $\Delta = n^{20/21}$, and the cost of the algorithm is therefore $O^*(n^{29/21})$. That is, we have

▶ **Theorem 4.** *Let P be a set of n points in $\mathbb{R}^3$, let s and t be two designated points, and let $0 < \lambda < n$ be a given parameter. Then we can find the smallest value r^* for which $G_{r^*}(P)$ contains a path between s and t that consists of at most λ edges, in $O^*(n^{29/21}) = O^*(n^{1.381})$ time.*

3.2 Higher dimensions

The algorithm given above can be generalized to points in $\mathbb{R}^d$, for any dimension $d \geq 4$. The geometric components in the algorithm are (i) constructing the grid, (ii) distance selection and the corresponding interval shrinking procedure, (iii) an efficient algorithm for the all-near-neighbor problem, (iv) an efficient algorithm for the all-nearest-neighbor problem, and, as a special case, (v) an efficient algorithm for the BCP problem.

The grid construction can trivially be extended to any fixed dimension. Distance selection in $\mathbb{R}^d$ takes $O^*(n^{2d/(d+1)})$ time (see, e.g., [3, Appendix]). The bipartite version, where we want to select distances between two given sets of m and n points, respectively, takes $O^*(m^{d/(d+1)}n^{d/(d+1)} + m + n)$ time, and the corresponding interval shrinking procedure, implemented using the technique of [9], takes $O^*(m^{d/(d+1)}n^{d/(d+1)}/L^{d/(d+1)})$ expected time, as long as m and n do not deviate significantly from one another. Agarwal and Matoušek [6] present an algorithm that solves problems (iii)–(v) for a pair of sets of respective sizes m and n, in $\mathbb{R}^d$ in time $O^*\left(m^{e/(e+1)}n^{e/(e+1)} + m + n\right)$, where $e = \lfloor d/2 \rfloor + 1$. (Recall that these problems are mapped, via the lifting transform given in Section 2, to dynamic halfspace range emptiness queries in $d+2$ dimensions, which explains the value of e; see [6].)

Substituting these components into the algorithm, with parameters L and Δ, the running time becomes

$$O^*\left(\frac{n\Delta^{(d-1)/(d+1)}}{L^{d/(d+1)}} + L^{1/2}n\Delta^{(e-1)/(e+1)} + n^{2e/(e+1)} + \frac{n^{(3e+1)/(e+1)}}{\Delta}\right),$$

where, as above, the third term is dominated by the fourth. We balance the terms in this bound, first by choosing L as a function of Δ, and then by choosing Δ as a function of n. As the exponents become rather messy, we simplify them somewhat by introducing another parameter $g = \frac{2(d-e)}{(e+1)(3d+1)}$. Straightforward, albeit rather tedious, calculations then yield

$$L = \Delta^{2g} \qquad \text{and} \qquad \Delta = n^{\frac{2e/(e+1)}{2e/(e+1)+g}},$$

and the bound then becomes

$$O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right).$$

As a sanity check, we verify that in $\mathbb{R}^3$, where $d = 3$, $e = 2$, and $g = 1/15$, the bound is indeed $O^*(n^{29/21})$. In conclusion, we have

▶ **Theorem 5.** *Let P be a set of n points in $\mathbb{R}^d$, for $d \geq 3$, let s and t be two designated points, and let $0 < \lambda < n$ be a given parameter. Then we can find the smallest value r^* for which $G_{r^*}(P)$ contains a path between s and t that consists of at most λ edges, in $O^*\left(n^{\frac{g}{2e/(e+1)+g}+\frac{2e}{e+1}}\right)$ randomized expected time, where $e = \lfloor d/2 \rfloor + 1$ and $g = \frac{2(d-e)}{(e+1)(3d+1)}$.*

As an additional example, in the case $d = 4$, with $e = 3$ and $g = 1/26$, the algorithm runs in $O^*(n^{61/40}) = O^*(n^{1.525})$ randomized expected time.

4 Reverse shortest paths in ball-intersection graphs in $\mathbb{R}^3$

As before, we use the shrink-and-bifurcate paradigm (see [7, 9, 11]). As we recall, the shrinking stage receives a number $L \ll n$ as input, and constructs an interval I that contains r^* and at most L other critical values of r. For arbitrary balls, the problem is represented in $\mathbb{R}^4$, where a ball with center c and radius ρ is mapped to the point (c, ρ). A value r is critical if two expanded balls $B_r(c_1, \rho_1) = B(c_1, \rho_1 + r)$ and $B_r(c_2, \rho_2) = B(c_2, \rho_2 + r)$ become externally tangent to each other, namely when

$$|c_1c_2| = (\rho_1 + r) + (\rho_2 + r), \qquad \text{or} \qquad r = \frac{1}{2}\left(|c_1c_2| - \rho_1 - \rho_2\right).$$

The improved shrinking procedure of [9] performs "distance selection" on various random samples from $\mathcal{B}$, where the selection has to find the k-th smallest critical value between pairs of balls in the cartesian product of the samples, for some given k. This selection process

is obtained in turn from a procedure that counts the number of critical values that are smaller than or equal to some given r_0. To do so, we rephrase the problem as an offline range searching problem, in which each ball $B = B(c_1, \rho_1)$ is mapped to the point (c_1, ρ_1) in $\mathbb{R}^4$, and also to the range[4]

$$\sigma_r(B) = \{(c, \rho) \in \mathbb{R}^4 \mid |cc_1| - \rho \leq 2r + \rho_1\}.$$

Each point has four degrees of freedom, and, for r fixed, so does each range. Using known results on semi-algebraic range searching (see [2] and [3, Appendix]), the cost of this procedure is $O^*(n^{8/5})$. In fact, running this "distance selection" procedure in a bipartite setting (which is the setting that arises when applying the technique of [9]), where we want to select critical values determined between pairs of balls in a pair of samples, consisting of m and n balls, respectively, the cost is $O^*(m^{4/5}n^{4/5} + m + n)$. Plugging this into the machinery of [9], the cost of the shrinking stage is $O^*(n^{8/5}/L^{4/5})$.

The cost of the bifurcation stage, as in all the previous applications, is $O^*(L^{1/2}D(n))$, where $D(n)$ is a bound on the running time of the decision procedure, which is our BFS algorithm on $G_r(\mathcal{B}) = G_0(\mathcal{B}_r)$, so $D(n) = O^*(n^{4/3})$. Hence the overall cost of the procedure is

$$O^*\left(\frac{n^{8/5}}{L^{4/5}} + L^{1/2}n^{4/3}\right).$$

We optimize this bound by choosing $L^{13/10} = n^{4/15}$, or $L = n^{8/39}$, and then the running time becomes $O^*(n^{56/39})$. That is, we have:

▶ **Theorem 6.** *The reverse shortest path problem on the ball-intersection graph of a set of n balls in $\mathbb{R}^3$ can be solved in $O^*(n^{56/39}) \approx O^*(n^{1.436})$ time.*

4.1 Higher dimensions

Here too, the machinery developed in Section 2 and the preceding part of Section 4 can be extended to any higher dimension d.

Breadth-First Search. We proceed as in the previous algorithm. The lifting transform used in the BFS implementation maps each ball $B_0 = B(c_0, \rho_0)$ to the point $B^* = (c_0, \rho_0, |c_0|^2 - \rho_0^2)$ in $\mathbb{R}^{d+2}$, and also to the halfspace

$$h_{B_0}: \qquad x_{d+2} \leq 2c_0 \cdot c + 2\rho_0\rho + \rho_0^2 - |c_0|^2$$

in $\mathbb{R}^{d+2}$. The dynamic halfspace range reporting structure of [6] yields, for any parameter $n \leq s \leq n^e$, where $e = \lfloor (d+2)/2 \rfloor = \lfloor d/2 \rfloor + 1$, a dynamic data structure of size $O^*(s)$, with initial construction cost $O^*(s)$, so that each query takes $O^*(n/s^{1/e})$ time, and each deletion takes $O^*(s/n)$ amortized time. Choosing $s = n^{2e/(e+1)}$, the size (and construction cost) becomes $O^*(s) = O^*(n^{2e/(e+1)})$, and each operation takes $O^*(n^{(e-1)/(e+1)})$ time, for a total of $O^*(n^{2e/(e+1)})$ time. Since this dominates the running time of the BFS, we obtain:

▶ **Theorem 7.** *BFS on the ball intersection graph of a set of n balls in $\mathbb{R}^d$ can be performed in $O^*(n^{2e/(e+1)})$ randomized expected time, where $e = \lfloor d/2 \rfloor + 1$.*

[4] Here the lifting transform to a halfspace in $\mathbb{R}^5$ does not seem to lead to a more efficient procedure. Technically, this is because the lifted problem to $\mathbb{R}^5$ now has to deal with halfspace range *counting*, whose performance is actually worse than the corresponding problem (involving semi-algebraic ranges instead of halfspaces) in $\mathbb{R}^4$.

Reverse Shortest Paths. Here the selection procedure, rephrased as an offline semi-algebraic range searching problem, involves points and ranges, each point and range with $d+1$ degrees of freedom. Hence the running time of this procedure, in the bipartite setting of sets of m and n (ball-representing) points, respectively, is (again, see [3, Appendix])

$$O^*\left(m^{(d+1)/(d+2)}n^{(d+1)/(d+2)} + m + n\right).$$

This implies, as in [9], that the cost of the interval shrinking procedure is

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

As in all its implementations, the bifurcation procedure runs in time

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

The overall running time is thus

$$O^*\left(n^{2(d+1)/(d+2)}/L^{(d+1)/(d+2)} + L^{1/2}n^{2e/(e+1)}\right).$$

Optimizing the value of L, we choose

$$L^{\frac{1}{2}+\frac{d+1}{d+2}} = n^{\frac{2(d+1)}{d+2}-\frac{2e}{e+1}}, \qquad \text{or} \qquad L = n^{\frac{4(d-e+1)}{(3d+4)(e+1)}},$$

and the overall running time of the algorithm is

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

(We verify that, for $d = 3$ and $e = 2$, this bound is indeed $O^*(n^{56/39})$.) That is, we have:

▶ **Theorem 8.** *The reverse shortest path problem on the ball intersection graph of a set of n balls in $\mathbb{R}^d$ can be solved in $O^*\left(n^{\frac{2(d+1)(3e+1)}{(3d+4)(e+1)}}\right)$ time, where $e = \lfloor d/2 \rfloor + 1$.*

(We remind the reader that the second improvement in [9] is not applicable here, when the balls do not have the same radius.)

For example, in $d = 4$ dimensions, with $e = 3$, the algorithm runs in $O^*(n^{25/16}) = O^*(n^{1.5625})$ time.

5 Further extensions

Other measures of expansion. The RSP technique can easily be applied to other well-behaved measures of expansion of the balls. For example, consider, in three dimensions, the case where, for $r \geq 1$, each ball $B(c, \rho)$ is expanded to the ball $B(c, r\rho)$ (all radii are multiplied by r). Now r is a critical value iff a pair $B(c_1, r\rho_1)$, $B(c_2, r\rho_2)$ of balls become externally tangent to each other, that is,

$$|c_1c_2| = r(\rho_1 + \rho_2), \qquad \text{or} \qquad r = \frac{|c_1c_2|}{\rho_1 + \rho_2}$$

Hence, in the selection procedure, we map each ball $B_0 = B(c_0, \rho_0)$ to the range

$$\sigma_{B_0} = \{(c, \rho) \in \mathbb{R}^4 \mid |cc_0| = r(\rho + \rho_0)\}.$$

Since these ranges are also semi-algebraic regions of constant complexity with four degrees of freedom, the same previously used offline semi-algebraic range searching machinery (on a different kind of ranges) can be applied here too, with the same asymptotic running time bound. The remainder of the procedure is carried out as before. Any other expansion rule can also be handled in this manner, as long as the corresponding ranges σ_{B_0} are semi-algebraic of constant complexity (with four degrees of freedom).

The same observations hold in any larger dimension d.

In conclusion, Theorems 4, 5, 6 and 8 continue to hold for any well-behaved measure of expansion.

References

1 A. Karim Abu-Affash, Paz Carmi, Matthew J. Katz, and Michael Segal. The Euclidean bottleneck Steiner path problem and other applications of (α, β)-pair decomposition. *Discret. Comput. Geom.*, 51(1):1–23, 2014. doi:10.1007/S00454-013-9550-9.

2 Pankaj 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.

3 Pankaj K. Agarwal, Boris Aronov, Esther Ezra, Matthew J. Katz, and Micha Sharir. Intersection queries for flat semi-algebraic objects in three dimensions and related problems. *ACM Trans. Algorithms*, 21(3):25:1–25:59, 2025. doi:10.1145/3721290.

4 Pankaj K. Agarwal, Haim Kaplan, Matthew J. Katz, and Micha Sharir. Segment proximity graphs and nearest neighbor queries amid disjoint segments. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 7:1–7:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.7.

5 Pankaj K. Agarwal, Matthew J. Katz, and Micha Sharir. On reverse shortest paths in geometric proximity graphs. *Comput. Geom.*, 117:102053, 2024. doi:10.1016/J.COMGEO.2023.102053.

6 Pankaj K. Agarwal and Jirí Matousek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995. doi:10.1007/BF01293483.

7 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. doi:10.1145/2700222.

8 Sergio Cabello and Miha Jejcic. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015. doi:10.1016/J.COMGEO.2014.12.003.

9 Timothy M. Chan and Zhengcheng Huang. Faster algorithms for reverse shortest path in unit-disk graphs and related geometric optimization problems: Improving the shrink-and-bifurcate technique. In *41st International Symposium on Computational Geometry, SoCG 2025*, volume 332 of *LIPIcs*, pages 32:1–32:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICS.SOCG.2025.32.

10 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.ISAAC.2016.24.

11 Haim Kaplan, Matthew J. Katz, Rachel Saban, and Micha Sharir. The unweighted and weighted reverse shortest path problem for disk graphs. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.67.

12 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.

13 Matthew J. Katz, Rachel Saban, and Micha Sharir. Near-linear algorithms for visibility graphs over a 1.5-dimensional terrain. In *32nd Annual European Symposium on Algorithms, ESA 2024*, volume 308 of *LIPIcs*, pages 77:1–77:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.77.

14 Katharina Klost. An algorithmic framework for the single source shortest path problem with applications to disk graphs. *Comput. Geom.*, 111:101979, 2023. doi:10.1016/J.COMGEO.2022.101979.

15 Jirí Matousek. Reporting points in halfspaces. *Comput. Geom.*, 2:169–186, 1992. doi:10.1016/0925-7721(92)90006-E.

16 Haitao Wang and Yiming Zhao. Reverse shortest path problem for unit-disk graphs. *J. Comput. Geom.*, 14(1):14–47, 2023. doi:10.20382/JOCG.V14I1A2.