Reverse Shortest Path Problem in Weighted Unit-Disk Graphs

Haitao Wang and Yiming Zhao(✉)

Department of Computer Science, Utah State University, Logan, UT 84322, USA
{haitao.wang,yiming.zhao}@usu.edu

Abstract. Given a set P of n points in the plane, a unit-disk graph $G_r(P)$ with respect to a parameter r is an undirected graph whose vertex set is P such that an edge connects two points $p, q \in P$ if the (Euclidean) distance between p and q is at most r (the weight of the edge is 1 in the unweighted case and is the distance between p and q in the weighted case). Given a value $\lambda > 0$ and two points s and t of P, we consider the following *reverse shortest path problem*: Compute the smallest r such that the shortest path length between s and t in $G_r(P)$ is at most λ. In this paper, we study the weighted case and present an $O(n^{5/4} \log^{5/2} n)$ time algorithm. We also consider the L_1 version of the problem where the distance of two points is measured by the L_1 metric; we solve the problem in $O(n \log^3 n)$ time for both the unweighted and weighted cases.

1 Introduction

Given a set P of n points in the plane and a parameter r, the *unit-disk graph* $G_r(P)$ is an undirected graph whose vertex set is P such that an edge connects two points $p, q \in P$ if the (Euclidean) distance between p and q is at most r. The weight of each edge of $G_r(P)$ is defined to be one in the *unweighted* case and is defined to the distance between the two vertices of the edge in the *weighted* case. Alternatively, $G_r(P)$ can be viewed as the intersection graph of the set of congruous disks centered at the points of P with radii equal to $r/2$, i.e., two vertices are connected if their disks intersect. The *length* of a path in $G_r(P)$ is the sum of the weights of the edges of the path.

Computing shortest paths in unit-disk graphs with different distance metrics and different weights assigning methods has been extensively studied, e.g., [5–7,12,13,17,19,20]. Although a unit-disk graph may have $\Omega(n^2)$ edges, geometric properties allow to solve the single-source-shortest-path problem (SSSP) in sub-quadratic time. Roditty and Segal [17] first proposed an algorithm of $O(n^{4/3+\epsilon})$ time for unit-disk graphs for both unweighted and weighted cases, for any $\epsilon > 0$. Cabello and Jejčič [5] gave an algorithm of $O(n \log n)$ time for the unweighted case. Using a dynamic data structure for bichromatic closest pairs [1], they also solved the weighted case in $O(n^{1+\epsilon})$ time [5]. Chan and Skrepetos [6] gave an

This research was supported in part by NSF under Grant CCF-2005323.

P. Mutzel et al. (Eds.): WALCOM 2022, LNCS 13174, pp. 135–146, 2022.
https://doi.org/10.1007/978-3-030-96731-4_12

$O(n)$ time algorithm for the unweighted case, assuming that all points of P are presorted. Kaplan et al. [13] developed a new randomized result for the dynamic bichromatic closest pair problem; applying the new result to the algorithm of [5] leads to an $O(n \log^{12+o(1)} n)$ expected time randomized algorithm for the weighted case. Recently, Wang and Xue [19] proposed a new algorithm that solves the weighted case in $O(n \log^2 n)$ time.

The L_1 version of the SSSP problem has also been studied, where the distance of two points in the plane is measured under the L_1 metric when defining $G_r(P)$. Note that in the L_1 version a "disk" is a diamond. The SSSP algorithms of [5,6] for the L_2 unweighted version can be easily adapted to the L_1 unweighted version. Wang and Zhao [20] recently solved the L_1 weighted case in $O(n \log n)$ time. It is known that $\Omega(n \log n)$ is a lower bound for the SSSP problem in both L_1 and L_2 versions [5,20]. Hence, the SSSP problem in the L_1 weighted/unweighted case as well as in the L_2 unweighted case has been solved optimally.

In this paper, we consider the following *reverse shortest path* (RSP) problem. In addition to P, given a value $\lambda > 0$ and two points $s, t \in P$, the problem is to find the smallest value r such that the distance between s and t in $G_r(P)$ is at most λ. Throughout the paper, we let r^* denote the optimal value r for the problem. The goal is therefore to compute r^*.

Observe that r^* must be equal to the distance of two points in P in any case (i.e., L_1, L_2, weighted, unweighted). For the L_2 unweighted case, Cabello and Jejčič [5] mentioned a straightforward solution that can solve it in $O(n^{4/3} \log^3 n)$ time, by using the distance selection algorithm of Katz and Sharir [14] to perform binary search on all interpoint distances of P; Wang and Zhao [21] later gave two algorithms with time complexities $O(\lfloor \lambda \rfloor \cdot n \log n)$ and $O(n^{5/4} \log^{7/4} n)$,[1] respectively, using the parametric search technique. The first algorithm is interesting for small λ and the second algorithm uses the first one as a subroutine.

In this paper, we study the L_2 weighted case of the RSP problem and present an algorithm of $O(n^{5/4} \log^{5/2} n)$ time. We also consider the L_1 version of the RSP problem and solve it in $O(n \log^3 n)$ time for both unweighted and weighted cases.

Recently, Katz and Sharir [15] proposed randomized algorithms of $O(n^{6/5+\epsilon})$ expected time for the L_2 RSP problem for both the unweighted and weighted cases, for arbitrary small $\epsilon > 0$.[2]

The RSP problem has been studied in the literature under various problem settings. Intuitively, the problem is to modify the graph (e.g., modify edge weights) so that certain desired constraints related to shortest paths can be satisfied, e.g., [4,22]. As a motivation of our problem, consider the following scenario. Suppose $G_r(P)$ represents a wireless sensor network in which each sensor is represented by a disk centered at a point in P and two sensors can communicate

[1] The time complexity given in [21] is $O(n^{5/4} \log^2 n)$, but can be easily improved to $O(n^{5/4} \log^{7/4} n)$ by changing the threshold for defining large cells from $n^{3/4}$ to $(n/\log n)^{3/4}$ in Sect. 4 [21].

[2] It is not explicitly stated in [15] that the algorithm is randomized. A key subroutine used in the algorithm is Theorem 1 [15], which is from [2] and is a randomized algorithm (see Sect. 4 in [2]).

with each other (e.g., directly transmit a message) if they are connected by an edge in $G_r(P)$. The disk radius is proportional to the energy of the sensor. The latency of transmitting a message between two neighboring sensors is proportional to their distance. For two sensors s and t, we want to know the minimum energy for all sensors so that the total latency of transmitting messages between s and t is no more than a target value λ. It is not difficult to see that this is equivalent to our RSP problem.

1.1 Our Approach

Our algorithm for the L_2 weighted RSP problem follows the parametric search scheme. Let $d_r(s,t)$ denote the distance from s to t in $G_r(P)$. Given any r, the *decision problem* is to decide whether $r^* \leq r$. Observe that $r^* \leq r$ holds if and only if $d_r(s,t) \leq \lambda$. Hence, the shortest path algorithm of Wang and Xue [19] (referred to the WX algorithm) can be used to solve the decision problem in $O(n \log^2 n)$ time. To compute r^*, since r^* is equal to the distance of two points of P, one could first compute all interpoint distances of points of P and then use the WX algorithm to perform binary search among these distances to compute r^*. Clearly, the algorithm takes $\Omega(n^2)$ time. Alternatively, as mentioned in [5], one can perform binary search by using the distance selection algorithm of Katz and Sharir [14] (i.e., given any k with $1 \leq k \leq \binom{n}{2}$, the algorithm finds the k-th smallest distance among all interpoint distances of P) without explicitly computing all these $\Omega(n^2)$ distances. As the algorithm of Katz and Sharir [14] runs in $O(n^{4/3} \log^2 n)$, this approach can compute r^* in $O(n^{4/3} \log^3 n)$ time.

We propose a more efficient parametric search algorithm, by "parameterizing" the decision algorithm, i.e., the WX algorithm. Like typical parametric search, we run the decision algorithm with a parameter $r \in (r_1, r_2]$ by simulating the decision algorithm on the unknown r^*. At each step, we call the decision algorithm on certain "critical values" r to compare r and r^*, and the algorithm will proceed accordingly based on the result of the comparison. The interval $(r_1, r_2]$ will also be shrunk after these comparisons but is guaranteed to contain r^* throughout the algorithm. The algorithm terminates once t is reached, at which moment we can prove that r^* is equal to r_2 of the current interval $(r_1, r_2]$.

For the L_1 RSP problem, we use an approach similar to the distance selection algorithm in [14]. As in the L_2 case, the decision problem can be solved in $O(n \log n)$ time by applying the SSSP algorithms for both the unweighted case and the weighted case [5,6,20,21] (more precisely, for the unweighted case, the decision problem can be solved in $O(n)$ time after $O(n \log n)$ time preprocessing for sorting the points of P [6]). Let Π denote the set of all pairwise distances of all points of P. In light of the observation that r^* is in Π, each iteration of our algorithm computes an interval $(a_j, b_j]$ (initially, $a_0 = -\infty$ and $b_0 = \infty$) such that $r^* \in (a_j, b_j]$ and the number of values of Π in $(a_j, b_j]$ is a constant fraction of the number of values of Π in $(a_{j-1}, b_{j-1}]$. In this way, r^* can be found within $O(\log n)$ iterations. Each iteration will call the decision algorithm to perform binary search on certain values. The total time of the algorithm is $O(n \log^3 n)$.

A by-product of our technique is an $O(n \log^3 n)$ time algorithm that can compute the k-th smallest L_1 distance among all pairs of points of P, for any given k with $1 \leq k \leq \binom{n}{2}$. As mentioned before, the L_2 version of the problem can be solved in $O(n^{4/3} \log^2 n)$ time [14].

Outline. In the following, we tackle the L_2 problem in Sects. 2. Due to the space limit, many proofs and the discussion about the L_1 problem are omitted but can be found in the full paper of [21] (the two papers are merged).

2 The L_2 RSP Problem

We follow the notation introduced in Sect. 1, e.g., P, $G_r(P)$, $d_r(s,t)$, r^*. Our goal is to compute r^*. As we will parameterize the WX algorithm, we first review the WX algorithm. For any two points p and q in the plane, let $\|p-q\|$ denote the Euclidean distance between them.

2.1 A Review of the WX Algorithm

Given P, r, and a source point $s \in P$, we consider the SSSP problem to compute shortest paths from s to all points of P in the unit-disk graph $G_r(P)$. The WX algorithm can solve the problem in $O(n \log^2 n)$ time.

For any point p, denote by $\odot_p$ the disk centered at p with radius r.

The first step is to implicitly build a grid $\Psi_r(P)$ of square cells whose side lengths are $r/2$. For simplicity of discussion, we assume that every point of P lies in the interior of a cell of $\Psi_r(P)$. A *patch* of $\Psi_r(P)$ refers to a square area consisting of 5×5 cells. For a point $p \in P$, we use $\square_p$ to denote the cell of $\Psi_r(P)$ containing p and use $\boxplus_p$ to denote the patch whose central cell is $\square_p$ (e.g., see Fig. 1). We refer to cells of $\boxplus_p \setminus \square_p$ as the *neighboring cells* of $\square_p$. As the side length of each cell of $\Psi_r(P)$ is $r/2$, any two points of P in a single cell of $\Psi_r(P)$ must be connected by an edge in $G_r(P)$. Moreover, if an edge connects two points p and q in $G_r(P)$, then q must lie in $\boxplus_p$ and vice versa. For any subset $Q \subseteq P$ and a cell $\square$ (resp.,a patch $\boxplus$) of $\Psi_r(P)$, define $Q_\square = Q \cap \square$ (resp., $Q_\boxplus = Q \cap \boxplus$). The step of implicitly building the grid actually computes the subset $P_\square$ for each cell $\square$ of $\Psi_r(P)$ that contains at least one point of P as well as associate pointers to each point $p \in P$ so that given any $p \in P$, the list of points of $P_{\square_p}$ (resp., $P_{\boxplus_p}$) can be accessed immediately. Building $\Psi_r(P)$ implicitly as above can be done in $O(n \log n)$ time and $O(n)$ space [19].

The WX algorithm follows the basic idea of Dijkstra's algorithm and computes an array $dist[\cdot]$ for each point $p \in P$, where $dist[p]$ will be equal to $d_r(s,p)$ when the algorithm terminates. Different from Dijkstra's shortest path algorithm, which picks a single vertex in each iteration to update the shortest path information of other adjacent vertices, the WX algorithm aims to update in each iteration the shortest path information for all points within one single cell of $\Psi_r(P)$ and pass on the shortest path information to vertices lying in the neighboring cells.

A key subroutine used in the WX algorithm is UPDATE(U, V), which updates the shortest path information for a subset $V \subseteq P$ of points by using the shortest path information of another subset $U \subseteq P$ of points. Specifically, the subroutine finds, for each $v \in V$, $q_v = \arg\min_{u \in U \cap \bigodot_v} \{dist[u] + \|u - v\|\}$ and update $dist[v] = \min\{dist[v], dist[q_v] + \|q_v - v\|\}$.

With the subroutine UPDATE(U, V), the WX algorithm works as follows.

Initially, we set $dist[s] = 0$, $dist[p] = \infty$ for all other points $p \in P \setminus \{s\}$, and $Q = P$. Then we enter the main (while) loop. In each iteration, we find a point z with minimum $dist$-value from Q, and then execute two update subroutines UPDATE$(Q_{\boxplus_z}, Q_{\square_z})$ and UPDATE$(Q_{\square_z}, Q_{\boxplus_z})$. Next, points of $Q_{\square_z}$ are removed from Q, because it can be shown that $dist[p]$ for all points $p \in Q_{\square_z}$ have been correctly computed [19]. The algorithm stops once Q becomes $\emptyset$. The efficiency of the algorithm hinges on the implementation of the two update subroutines. We give some details below, which are needed in our RSP algorithm as well.

The First Update. For the first update UPDATE$(Q_{\boxplus_z}, Q_{\square_z})$, the crucial step is finding a point $q_v \in Q_{\boxplus_z} \cap \bigodot_v$ for each point $v \in Q_{\square_z}$ such that $dist[q_v] + \|q_v - v\|$ is minimized. If we assign $dist[q]$ as a weight to each point $q \in Q_{\boxplus_z}$, then the problem is equivalent to finding the additively-weighted nearest neighbor q_v from $Q_{\boxplus_z} \cap \bigodot_v$ for each $v \in Q_{\square_z}$. To this end, Wang and Xue [19] proved a *key observation* that any point $q \in Q_{\boxplus_z}$ that minimizes $dist[q] + \|q - v\|$ must lie in $\bigodot_v$. This implies that for each point $v \in Q_{\square_z}$, its additively-weighted nearest neighbor in $Q_{\boxplus_z}$ is also its additively-weighted nearest neighbor in $Q_{\boxplus_z} \cap \bigodot_v$. As such, q_v for all $v \in Q_{\square_z}$ can be found by first building an additively-weighted Voronoi Diagram on points of $Q_{\boxplus_z}$ [9] and then performing point locations for all $v \in Q_{\square_z}$ [8,16,18]. In this way, since $\sum_{z_i} |P_{\boxplus_{z_i}}| = O(n)$, where z_i refers to the point z in the i-th iteration of the main loop, the first updates for all iterations of the main loop can be done in $O(n \log n)$ time in total [19].

The Second Update. The second update UPDATE$(Q_{\square_z}, Q_{\boxplus_z})$ is more challenging because the above key observation no longer holds. Since $Q_{\boxplus_z}$ has $O(1)$ cells of $\Psi_r(P)$, it suffices to perform UPDATE$(Q_{\square_z}, Q_{\square})$ for all cells $\square \in \boxplus_z$.

If $\square$ is $\square_z$, then $Q_{\square_z} = Q_{\square}$. Since the distance between any two points in $\square_z$ is at most r, we can easily implement UPDATE$(Q_{\square_z}, Q_{\square})$ in $O(|Q_{\square_z}| \log |Q_{\square_z}|)$ time, by first building a additively-weighted Voronoi diagram on points of $Q_{\square_z}$ (each point $q \in Q_{\square_z}$ is assigned a weight equal to $dist[q]$), and then using it to find the additively-weighted nearest neighbor q_v for each point $v \in Q_{\square_z}$.

If $\square$ is not $\square_z$, a useful property is that $\square$ and $\square_z$ are separated by an axis-parallel line. The WX algorithm implements UPDATE$(Q_{\square_z}, Q_{\square})$ with the following three steps. Let $U = Q_{\square_z}$ and $V = Q_{\square}$.

1. Sort points of U as $\{u_1, u_2, ..., u_{|U|}\}$ such that $dist[u_1] \leq dist[u_2] \leq ... \leq dist[u_{|U|}]$.
2. Compute $|U|$ disjoint subsets $\{V_1, ..., V_{|U|}\}$ with $V_i = \{v \in V \mid v \in \bigodot_{u_i}$ and $v \notin \bigodot_{u_j}$ for all $1 \leq j < i\}$. Equivalently, for each point $v \in V$, v is in V_{i_v}, where i_v is the smallest index i (if exists) such that $\bigodot_{u_i}$ contains v.

3. Initialize $U' = \emptyset$. Proceed with $|U|$ iterations for $i = |U|, |U|-1, ..., 1$ sequentially and do the following in each iteration for i: (1) Add u_i to U'; (2) for each point $v \in V_i$, compute $q_v = \arg\min_{u \in U'}\{dist[u] + \|u - v\|\}$; (3) update $dist[v] = \min\{dist[v], dist[q_v] + \|q_v - v\|\}$.

By the definition of V_i, $U \cap \bigodot_v \subseteq U' = \{u_{|U|}, u_{|U|-1}, ..., u_i\}$ for each $v \in V_i$ in the iteration for i of Step 3. Wang and Xue [19] proved that q_v found for each $v \in V_i$ in Step 3 must lie in $\bigodot_v$. They gave a method to implement Step 2 in $O(k \log k)$ time by making use of the property that U and V are separated by an axis-parallel line, where $k = |U|+|V|$. Step 3 can be considered as an offline insertion-only additively-weighted nearest neighbor searching problem and the WX algorithm solves the problem in $O(k \log^2 k)$ time using the standard logarithmic method [3], with $k = |U| + |V|$.

As such, the second updates for all iterations in the WX algorithm takes $O(n \log^2 n)$ time in total [19], which dominates the entire algorithm (other parts of the algorithm together takes $O(n \log n)$ time).

2.2 The RSP Algorithm

We now tackle the RSP problem, i.e., computing r^* for two points $s, t \in P$ and a value λ, by "parameterizing" the WX algorithm.

Recall that the decision problem is to decide whether $r^* \leq r$ for a given r. Notice that $r^* \leq r$ holds if and only if $d_r(s,t) \leq \lambda$. The decision problem can be solved in $O(n \log^2 n)$ time by running the WX algorithm on r. In the following, we refer to the WX algorithm as the *decision algorithm*. We say that r is a *feasible value* if $r^* \leq r$ and an *infeasible value* otherwise.

As discussed in Sect. 1, to find r^*, we run the decision algorithm with a parameter r in an interval $(r_1, r_2]$ by simulating the algorithm on the unknown r^*. The interval always contains r^* but will be shrunk during course of the algorithm (for simplicity, when we say $(r_1, r_2]$ is shrunk, this also include the case that $(r_1, r_2]$ does not change). Initially, we set $r_1 = 0$ and $r_2 = \infty$.

The first step is to build a grid for P. The goal is to shrink $(r_1, r_2]$ so that it contains r^* and if $r^* \neq r_2$ (and thus $r^* \in (r_1, r_2)$), for any $r \in (r_1, r_2)$, the grid $\Psi_r(P)$ has the same combinatorial structure as $\Psi_{r^*}(P)$ in the following sense: (1) Both grids have the same number of rows and columns; (2) for any point $p \in P$, p lies in the i-th row and j-th column of $\Psi_r(P)$ if and only if p lies in the i-th row and j-th column of $\Psi_{r^*}(P)$. This step is also needed in the algorithm of [21] for solving the unweighted case of the RSP problem and an $O(n \log n)$ time algorithm was given in [21] to achieve this by using the sorted matrix searching technique [10,11] along with the linear-time decision algorithm for the unweighted case [6] (more specifically, the decision algorithm is called $O(\log n)$ times). Here in our weighted problem, we can apply exactly the same algorithm except that we use our $O(n \log^2 n)$ time decision algorithm instead and the total time thus becomes $O(n \log^3 n)$.

Let $(r_1, r_2]$ denote the interval after building the grid. We pick any $r \in (r_1, r_2)$ and call the WX algorithm on r to compute a grid $\Psi_r(P)$. Recall from Sect. 2.1 that by "computing $\Psi_r(P)$", we mean to compute the following *grid information*: $P_\square$ for each cell $\square$ of $\Psi_r(P)$ that contains at least one point of P as well as the associated information (e.g., for finding cells of $P_{\boxplus_p}$). These information is the same as that of $\Psi_{r^*}(P)$ if $r^* \neq r_2$. Below, we will simply use $\Psi(P)$ to refer to the grid information computed above, meaning that it does not change with respect to $r \in (r_1, r_2)$.

We use $dist_r[\cdot]$, $Q(r)$, $z(r)$ respectively to refer to $dist[\cdot]$, Q, z in the WX algorithm running on a parameter r. We start with setting $dist_r[s] = 0$, $dist_r[p] = \infty$ for all $p \in P \setminus \{s\}$, and $Q(r) = P$.

Next we enter the main loop. As long as $Q(r) \neq \emptyset$, each iteration finds a point $z(r)$ with the minimum $dist_r$-value from $Q(r)$ and update $dist_r$-values for points in $Q(r)_{\square_{z(r)}} \cup Q(r)_{\boxplus_{z(r)}}$. Points in $Q(r)_{\square_{z(r)}}$ are then removed from $Q(r)$. Each iteration will shrink $(r_1, r_2]$ such that the following algorithm invariant is maintained: $(r_1, r_2]$ contains r^* and if $r^* \neq r_2$, the following holds for all $r \in (r_1, r_2)$: $z(r) = z(r^*)$, $Q(r) = Q(r^*)$, and $dist_r[p] = dist_{r^*}[p]$ for all $p \in P$.

Consider an iteration of the main loop. We assume that the invariant holds before the iteration on the interval $(r_1, r_2]$, which is true before the first iteration. In the following, we describe our algorithm for the iteration and we will show that the invariant holds after the iteration. We assume that $r^* \neq r_2$. According to our invariant, for any $r \in (r_1, r_2)$, we have $z(r) = z(r^*)$, $Q(r) = Q(r^*)$, and $dist_r[p] = dist_{r^*}[p]$ for all $p \in P$.

We first find a point $z(r) \in Q(r)$ with the minimum $dist_r$-value. Since the invariant holds before the iteration, we have $z(r) = \arg\min_{p \in Q(r)} dist_r[p] = \arg\min_{p \in Q(r^*)} dist_{r^*}[p] = z(r^*)$. If ties happen, we follow the same way as the WX algorithm to break ties and ensure $z(r) = z(r^*)$. Hence, no "parameterization" is needed in this step, i.e., all involved values in the computation of this step are independent of r.

Next, we perform the first update $\textsc{Update}(Q(r)_{\boxplus_{z(r)}}, Q(r)_{\square_{z(r)}})$. This step also does not need parameterization. Indeed, for each point $p \in Q(r)_{\boxplus_{z(r)}}$, we assign $dist_r[p]$ to p as a weight, and then construct the additively-weighted Voronoi diagram on $Q(r)_{\boxplus_{z(r)}}$. For each point $v \in Q(r)_{\square_{z(r)}}$, we use the diagram to find its additively-weighted nearest neighbor $q_v(r) \in Q(r)_{\boxplus_{z(r)}}$ and update $dist_r[v] = \min\{dist_r[v], dist_r[q_v(r)] + \|q_v(r) - v\|\}$. Since $z(r) = z(r^*)$, and $Q(r) = Q(r^*)$, we have $Q(r)_{\boxplus_{z(r)}} = Q(r^*)_{\boxplus_{z(r^*)}}$ and $Q(r)_{\square_{z(r)}} = Q(r^*)_{\square_{z(r^*)}}$. Further, since $dist_r[p] = dist_{r^*}[p]$ for all $p \in P$, for each point $v \in Q(r)_{\square_{z(r)}}$, $q_v(r) = q_v(r^*)$ and each updated $dist_r[v]$ in our algorithm is equal to the corresponding updated $dist_{r^*}[v]$ in the same iteration of the WX algorithm running on r^*. As such, the invariant still holds after the first update.

Implementing the second update $\textsc{Update}(Q(r)_{\square_{z(r)}}, Q(r)_{\boxplus_{z(r)}})$ is more challenging and parameterization is necessary. It suffices to implement the updates $\textsc{Update}(Q(r)_{\square_{z(r)}}, Q(r)_\square)$ for all cells $\square \in \boxplus_{z(r)}$.

If $\square$ is $\square_{z(r)}$, then $Q(r)_{\square_{z(r)}} = Q(r)_\square$. In this case, again no parameterization is needed. Since the distance between any two points in $\square_{z(r)}$ is at most r, we can easily implement $\textsc{Update}(Q(r)_{\square_{z(r)}}, Q(r)_\square)$ in $O(|Q(r)_{\square_{z(r)}}| \log |Q(r)_{\square_{z(r)}}|)$ time,

by first building an additively-weighted Voronoi diagram on points of $Q(r)_{\square_{z(r)}}$ (each point $p \in Q(r)_{\square_{z(r)}}$ is assigned a weight equal to $dist_r[p]$), and then using it to find the additively-weighted nearest neighbor $q_v(r)$ for each point $v \in Q(r)_{\square_z}$. By an analysis similar to the above first update, the invariant still holds.

We now consider the case where $\square$ is not $\square_{z(r)}$. In this case, $\square$ and $\square_{z(r)}$ are separated by an axis-parallel line ℓ. Without loss of generality, we assume that ℓ is horizontal and $\square_{z(r)}$ is below ℓ. Since $z(r) = z(r^*)$ and $Q(r) = Q(r^*)$ for all $r \in (r_1, r_2)$, we let $U = Q(r)_{\square_{z(r)}}$ and $V = Q(r)_{\square}$, meaning that both U and V are independent of $r \in (r_1, r_2)$. Recall that there are three steps in the second update of the decision algorithm. Our algorithm needs to simulate all three steps. As will be seen later, only the second step needs parameterization.

The first step is to sort points in U by their $dist_r$-values. Since $dist_r[p] = dist_{r^*}[p]$ for all $p \in P$, the sorted list $\{u_1, u_2, ..., u_{|U|}\}$ of U obtained in our algorithm is the same as that obtained in the decision algorithm running on r^*.

For any r, denote by $\bigodot_p(r)$ the disk centered at a point p with radius r.

The second step is to compute $|U|$ disjoint subsets $\{V_1(r), V_2(r), ..., V_{|U|}(r)\}$ of V such that $V_i(r) = \{v \mid i_v(r) = i, v \in V\}$, where $i_v(r)$ is the smallest index such that $\bigodot_{u_{i_v(r)}}(r)$ contains point v. This step needs parameterization. We will shrink the interval $(r_1, r_2]$ so that it still contains r^* and if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, $V_i(r) = V_i(r^*)$ holds for all $1 \leq i \leq |U|$ (it suffices to ensure $i_v(r) = i_v(r^*)$ for all $v \in V$). Our algorithm relies on the following observation, which is based on the definition of $i_v(r)$.

Observation 1. *For any point $v \in V$, if $\bigodot_{u_j}(r)$ contains v with $1 \leq j \leq |U|$, then $i_v(r) \leq j$.*

For a subset $P' \subseteq P$, let $\mathcal{F}_r(P')$ denote the union of the disks centered at points of P' with radius r. We first solve a subproblem in the following lemma.

Lemma 1. *Suppose $(r_1, r_2]$ contains r^* such that if $r^* \neq r_2$, then for all $r \in (r_1, r_2)$, $dist_r[p] = dist_{r^*}[p]$ for all points $p \in P$. For a subset $U' \subseteq U$ and a subset $V' \subseteq V$, in $O(n \log^2 n \cdot \log(|U'| + |V'|))$ time we can shrink $(r_1, r_2]$ so that it still contains r^* and if $r^* \neq r_2$, then for all $r \in (r_1, r_2)$, for any $v \in V'$, v is contained in $\mathcal{F}_r(U')$ if and only if v is contained in $\mathcal{F}_{r^*}(U')$.*

Recall that we have an interval $(r_1, r_2]$. Our goal is to shrink it so that it still contains r^* and if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, $V_i(r) = V_i(r^*)$ holds for all $1 \leq i \leq |U|$. With Observation 1 and Lemma 1, we have the following lemma.

Lemma 2. *We can shrink the interval $(r_1, r_2]$ in $O(n \log^4 n)$ time so that it still contains r^* and if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, $V_i(r) = V_i(r^*)$ holds for all $1 \leq i \leq |U|$.*

Proof. To have $V_i(r) = V_i(r^*)$ for all $1 \leq i \leq |U|$, it suffices to ensure $i_v(r) = i_v(r^*)$ for all points $v \in V$. Let $M = |U|$ and $N = |V|$. Note that $M \leq n$ and $N \leq n$.

As defined in the proof of Lemma 1, for any subset $U' \subseteq U$ and any r, denote by $\mathcal{U}_r(U')$ the upper envelope of the portions of $\bigodot_u(r)$ above ℓ for all $u \in U'$.

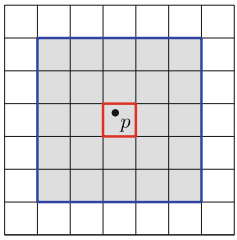

Fig. 1. The red cell that contains the point p is $\square_p$ and the square area bounded by blue segments is the patch $\boxplus_p$. All adjacent vertices of p in $G_r(P)$ must lie in the grey region. (Color figure online)

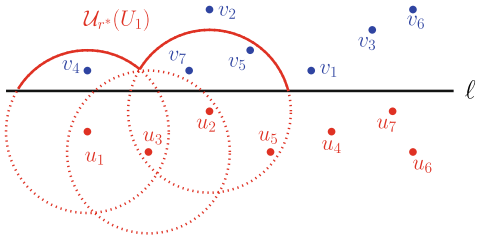

Fig. 2. Illustrating U_1 and V_1, where $U_1 = \{u_1, u_2, u_3\}$ and $V_1 = \{v_4, v_5, v_7\}$. The solid arcs are on $\mathcal{U}_{r^*}(U_1)$.

In light of Observation 1, we use the divide and conquer approach. Recall that $U = \{u_1, u_2, \ldots, u_M\}$. Consider the following subproblem on (U, V): shrink $(r_1, r_2]$ so that it still contains r^* and if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, for any $v \in V$, v is below $\mathcal{U}_r(U_1)$ if and only if v is below $\mathcal{U}_{r^*}(U_1)$, where U_1 is the first half of U, i.e., $U_1 = \{u_1, u_2, ..., u_{\lfloor \frac{M}{2} \rfloor}\}$. The subproblem can be solved in $O(n \log^3 n)$ time by applying Lemma 1. Next, we pick any $r \in (r_1, r_2)$ and compute $\mathcal{U}_r(U_1)$ and find the subset V_1 of the points of V that are below $\mathcal{U}_r(U_1)$ (e.g., see Fig. 2). By Observation 1, for each point $v \in V$, $i_v(r) \leq \lfloor \frac{M}{2} \rfloor$ if $v \in V_1$ and $i_v(r) > \lfloor \frac{M}{2} \rfloor$ otherwise. By the above property of $(r_1, r_2]$, for each point $v \in V$, we also have $i_v(r^*) \leq \lfloor \frac{M}{2} \rfloor$ if $v \in V_1$ and $i_v(r^*) > \lfloor \frac{M}{2} \rfloor$ otherwise.

Next, we solve two subproblems recursively: one on (U_1, V_1) and the other on $(U \setminus U_1, V \setminus V_1)$. Both subproblems use $(r_1, r_2]$ as their "input intervals" and solving each subproblem will produce a shrunk "output interval" $(r_1, r_2]$. Consider a subproblem on (U', V') with $U' \subseteq U$ and $V' \subseteq V$. If $|U'| = 1$, then we solve the problem "directly" (i.e., this is the base case) as follows. Assume that $r^* \neq r_2$ and let r be any value in (r_1, r_2). Let u_j be the only point of U'. If $j < M$, according to our algorithm and based on Observation 1, $i_v(r) = i_v(r^*) = j$ holds for all points $v \in V'$. If $j = M$, however, for each point $v \in V'$, it is possible that v is not contained in $\bigodot_u(r^*)$ for any point $u \in U$, in which case v is not below $\mathcal{U}_{r^*}(U)$ and thus is not below $\mathcal{U}_{r^*}(U')$. On the other hand, if v is below $\mathcal{U}_{r^*}(U')$, then $i_v(r^*) = M$. To solve the problem, we can simply apply Lemma 1 on U' and V', after which we obtain an interval $(r_1, r_2]$. Then, we pick any $r \in (r_1, r_2)$ and for any $v \in V'$ with v contained in $\bigodot_{u_M}(r)$, $i_v(r) = i_v(r^*) = M$ holds if $r^* \neq r_2$.

The above divide-and-conquer algorithm can be viewed as a binary tree structure T in which each node represents a subproblem. Clearly, the height of T is $O(\log M)$ and T has $\Theta(M)$ nodes. If we solve each subproblem individually by Lemma 1 as described above, then the algorithm would take $\Omega(Mn)$ time because there are $\Omega(M)$ subproblems and solving each subproblem by Lemma 1

takes $\Omega(n)$ time, which would result in an $\Omega(n^2)$ time algorithm in the worst case. To reduce the runtime, instead, we solve subproblems at the same level of T simultaneously (or "in parallel") by applying the algorithm of Lemma 1. We can show that solving all subproblems in the same level of T can be done in $O(n \log^3 n)$ time. The details are given in our full paper. As T has $O(\log M)$ levels, the total time of the overall algorithm is $O(n \log^4 n)$. □

With Lemma 2, we obtain subsets $\{V_1(r), V_2(r), ..., V_{|U|}(r)\}$ and an interval $(r_1, r_2]$ containing r^* such that if $r^* \neq r_2$, for any $r \in (r_1, r_2)$, $V_i(r) = V_i(r^*)$ holds for all $1 \leq i \leq |U|$. Note that neither the array $dist_r[\cdot]$ nor $Q(r)$ is modified during the algorithm of Lemma 2. Hence, if $r^* \neq r_2$, for all $r \in (r_1, r_2]$, we still have $Q(r) = Q(r^*)$ and $dist_r[p] = dist_{r^*}[p]$ for all points $p \in P$. Thus, our algorithm invariant still holds. This finishes the second step of the second update.

The third step of the second update is to solve the offline insertion-only additively-weighted nearest neighbor searching problem. This step does not need parameterization. Similar to the first update, we pick any $r \in (r_1, r_2)$ and apply the WX algorithm directly. Indeed, the algorithm on r^* only relies on the following information: U and its sorted list by $dist_{r^*}[\cdot]$ values and the subsets $V_1(r^*), \ldots, V_{|U|}(r^*)$. Recall that if $r^* \neq r_2$, then for all $r \in (r_1, r_2)$, $dist_r[p] = dist_{r^*}[p]$ for all $p \in P$, and $V_i(r) = V_i(r^*)$ for all $1 \leq i \leq |U|$. As such, if we pick any $r \in (r_1, r_2)$ and apply the WX algorithm directly, $dist_r[v] = dist_{r^*}[v]$ holds for all points $v \in V$ after this step. Therefore, as in the WX algorithm, this step can be done in $O(k \log^2 k)$ time, where $k = |U| + |V|$.

This finishes the second update of the algorithm. As discussed above, the algorithm invariant holds for the interval $(r_1, r_2]$.

The final step of the iteration is to remove points in $Q(r)_{\square_{z(r)}}$ from $Q(r)$. Since if $r^* \neq r_2$, for all $r \in (r_1, r_2)$, $Q(r) = Q(r^*)$, $z(r) = z(r^*)$, and $Q(r)_{\square_{z(r)}} = Q(r^*)_{\square_{z(r^*)}}$, $Q(r) = Q(r^*)$ still holds after this point removal operation. Therefore, our algorithm invariant holds after the iteration.

In summary, each iteration of our algorithm takes $O(n \log^4 n)$ time. If the point t is contained in $\square_{z(r)}$ (i.e., t is reached) in the current iteration, then we terminate the algorithm. The following lemma shows that we can simply return r_2 as r^*.

Lemma 3. *Suppose that t is contained in $\square_{z(r)}$ in an iteration of our algorithm and $(r_1, r_2]$ is the interval after the iteration. Then $r^* = r_2$.*

The algorithm may take $\Omega(n^2)$ time because t may be reached in $\Omega(n)$ iterations. A further improvement is discussed in the next subsection.

2.3 A Further Improvement

To further reduce the runtime of the algorithm, we borrow a technique from [21] to partition the cells of the grid into large and small cells.

As before, we first compute the grid information $\Psi(P)$ and obtain an interval $(r_1, r_2]$. Let $\mathcal{C}$ denote the set of all non-empty cells of $\Psi(P)$ (i.e., cells that contain

at least one point of P). For each cell $C \in \mathcal{C}$, let $N(C)$ denote the set of non-empty neighboring cells of C in $\mathcal{C}$ and $P(C)$ the set of points of P contained in cell C. We have $|N(C)| = O(1)$ and $|\mathcal{C}| = O(n)$. A cell C of $\mathcal{C}$ is a *large cell* if it contains at least $n^{3/4}\log^{3/2} n$ points of P, i.e., $|P(C)| \geq n^{3/4}\log^{3/2} n$, and a *small cell* otherwise. Clearly, $\mathcal{C}$ has at most $n^{1/4}/\log^{3/2} n$ large cells. For all pairs of non-empty neighboring cells (C, C'), with $C \in \mathcal{C}$ and $C' \in N(C)$, (C, C') is a *small-cell pair* if both C and C' are small cells, and a *large-cell pair* otherwise, i.e., at least one cell is a large cell. Since $N(C) = O(1)$ for each cell $C \in \mathcal{C}$, there are $O(n^{1/4}/\log^{3/2} n)$ large-cell pairs.

We first provide some intuition about our approach and then fresh out the details. Notice that in each iteration of the main loop in our previous algorithm, only the second step of the second update parameterizes the WX algorithm (i.e., the decision algorithm is called on certain critical values); in that step, we need to process $O(1)$ pairs of cells (C, C') with $C \in \mathcal{C}$ and $C' \in N(C)$. No matter how many points of P contained in the two cells, we need $O(n\log^4 n)$ time to perform the parametric search due to Lemma 2. To reduce the time, we preprocess all small-cell pairs so that the algorithm only needs to perform the parametric search for large-cell pairs. Since there are only $O(n^{1/4}/\log^{3/2} n)$ large-cell pairs, the total time we spend on parametric search can be reduced to $O(n^{5/4}\log^{5/2} n)$. For those small-cell pairs, the preprocessing provides sufficient information to allow us to simply run the original WX algorithm without resorting to parametric search. Specifically, before we enter the main loop of the algorithm (and after the grid information $\Psi(P)$ is computed, along with an interval $(r_1, r_2]$), we preprocess all small-cell pairs using the following lemma which is similar to [21].

Lemma 4. *In $O(n^{5/4}\log^{5/2} n)$ time we can shrink the interval $(r_1, r_2]$ so that it still contains r^* and if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, for any small-cell pair (C, C') with $C \in \mathcal{C}$ and $C' \in N(C)$, an edge connects a point $p \in P(C)$ and a point $p' \in P(C')$ in $G_r(P)$ if and only if an edge connects p and p' in $G_{r^*}(P)$.*

Let $(r_1, r_2]$ denote the interval obtained after the preprocessing for all small-cell pairs in Lemma 4. Lemma 4 essentially guarantees that if $r^* \neq r_2$, then for any $r \in (r_1, r_2)$, the adjacency relation of points in any small-cell pair in $G_r(P)$ is the same as that in $G_{r^*}(P)$. Note that if $(r_1, r_2]$ is shrunk so that it still contains r^*, then the above property still holds for the shrunk interval. Based on this property, combining with our previous algorithm, we have the following theorem.

Theorem 1. *The reverse shortest path problem for unit-disk graphs in the L_2 weighted case can be solved in $O(n^{5/4}\log^{5/2} n)$ time.*

References

1. Agarwal, P., Efrat, A., Sharir, M.: Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. SIAM J. Comput. **29**, 912–953 (1999)

2. Avraham, R.B., Filtser, O., Kaplan, H., Katz, M.J., Sharir, M.: The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. ACM Trans. Algorithms **11**(4), 1–29 (2015). Article No. 29
3. Bentley, J.: Decomposable searching problems. Inf. Process. Lett. **8**, 244–251 (1979)
4. Burton, D., Toint, P.: On an instance of the inverse shortest paths problem. Math. Program. **53**, 45–61 (1992)
5. Cabello, S., Ejčič, M.: Shortest paths in intersection graphs of unit disks. Comput. Geom. Theory Appl. **48**(4), 360–367 (2015)
6. Chan, T., Skrepetos, D.: All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In: Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC), pp. 24:1–24:13 (2016)
7. Chan, T., Skrepetos, D.: Approximate shortest paths and distance oracles in weighted unit-disk graphs. In: Proceedings of the 34th International Symposium on Computational Geometry (SoCG), pp. 24:1–24:13 (2018)
8. Edelsbrunner, H., Guibas, L., Stolfi, J.: Optimal point location in a monotone subdivision. SIAM J. Comput. **15**(2), 317–340 (1986)
9. Fortune, S.: A sweepline algorithm for Voronoi diagrams. Algorithmica **2**, 153–174 (1987). https://doi.org/10.1007/BF01840357
10. Frederickson, G., Johnson, D.: Generalized selection and ranking: sorted matrices. SIAM J. Comput. **13**(1), 14–30 (1984)
11. Frederickson, G., Johnson, D.: Finding kth paths and p-centers by generating and searching good data structures. J. Algorithms **4**(1), 61–80 (1983)
12. Gao, J., Zhang, L.: Well-separated pair decomposition for the unit-disk graph metric and its applications. SIAM J. Comput. **35**(1), 151–169 (2005)
13. Kaplan, H., Mulzer, W., Roditty, L., Seiferth, P., Sharir, M.: Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 2495–2504 (2017)
14. Katz, M., Sharir, M.: An expander-based approach to geometric optimization. SIAM J. Comput. **26**(5), 1384–1408 (1997)
15. Katz, M.J., Sharir, M.: Efficient algorithms for optimization problems involving distances in a point set. arXiv:2111.02052 (2021)
16. Kirkpatrick, D.: Optimal search in planar subdivisions. SIAM J. Comput. **12**(1), 28–35 (1983)
17. Roditty, L., Segal, M.: On bounded leg shortest paths problems. Algorithmica **59**(4), 583–600 (2011)
18. Sarnak, N., Tarjan, R.: Planar point location using persistent search trees. Commun. ACM **29**, 669–679 (1986)
19. Wang, H., Xue, J.: Near-optimal algorithms for shortest paths in weighted unit-disk graphs. Discret. Comput. Geom. **64**, 1141–1166 (2020)
20. Wang, H., Zhao, Y.: An optimal algorithm for L_1 shortest paths in unit-disk graphs. In: Proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG), pp. 211–218 (2021)
21. Wang, H., Zhao, Y.: Reverse shortest path problem for unit-disk graphs. In: Proceedings of the 17th International Symposium of Algorithms and Data Structures (WADS), pp. 655–668 (2021). https://arxiv.org/abs/2104.14476
22. Zhang, J., Lin, Y.: Computation of the reverse shortest-path problem. J. Global Optim. **25**(3), 243–261 (2003)