Reverse Shortest Path Problem in Weighted Unit-Disk Graphs*

Haitao Wang† Yiming Zhao‡

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. The unweighted case of the problem was solved in $O(n^{5/4}\log^{7/4} n)$ time before. In this abstract, 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 in the plane is measured by the L_1 metric. We solve the L_1 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., [3–7, 9–11]. 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 [9] 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 [3] 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 [3]. Chan and Skrepetos [4] gave an $O(n)$ time algorithm for the unweighted case, assuming that all points of P are presorted. Kaplan et al. [7] developed a new randomized result for the dynamic bichromatic closest pair problem; applying the new result to the algorithm of [3] leads to an $O(n\log^{12+o(1)} n)$ expected time randomized algorithm for the weighted case. Recently, Wang and Xue [10] 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 the graph $G_r(P)$. Note that in the L_1 version a "disk" becomes a diamond. The SSSP algorithms of [3, 4] for the L_2 unweighted version can be easily

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

†Department of Computer Science, Utah State University, Logan, UT 84322, USA. haitao.wang@usu.edu

‡Corresponding author. Department of Computer Science, Utah State University, Logan, UT 84322, USA. yiming.zhao@usu.edu

adapted to the L_1 unweighted version. Wang and Zhao [11] recently solved the L_1 weighted case in $O(n \log n)$ time. It is also known $\Omega(n \log n)$ is a lower bound for the SSSP problem in both L_1 and L_2 versions [3, 11]. 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 abstract, 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 abstract, 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 [3] 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 [8] to perform binary search on all interpoint distances of P; Wang and Zhao [12] 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. In particular, the first algorithm is interesting when λ is relatively small and the second algorithm uses the first one as a subroutine.

In this abstract, 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. In addition, we also consider the L_1 version and solve the L_1 RSP problem in $O(n \log^3 n)$ time for both the unweighted and weighted cases.

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., [2, 13]. 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 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.

2 Our algorithms – an overview

In this section, we give a brief overview on our algorithms. We begin with the L_2 weighted RSP problem.

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 [10] (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 [3], one can perform binary search by using the distance selection algorithm of Katz and Sharir [8] (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 [8] 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 algorithms, we run the decision algorithm with a parameter $r \in (r_1, r_2]$ by simulating the decision algorithm on the unknown r^*. At

[1]The time complexity given in [12] 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 Section 4 [12].

each step of the algorithm, 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 $(r_1, r_2]$.

Specifically, the WX algorithm first builds a grid $\Psi_r(P)$ implicitly on the plane such that for any point $p \in P$, if p is in a cell C of the grid, then all neighboring points of p in $G_r(P)$ lie in a constant number of neighboring cells of C. 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. More specifically, each iteration of the algorithm picks a vertex z with the minimum $dist$-value. We assume that z lies in cell C of $\Psi_r(P)$. We update the shortest information ($dist$-values) of vertices lying in cell C and vertices lying in a constant number of neighboring cells of C. After that, it can be proved that $dist[p] = d_r(s, p)$ for all points p of P in C [10].

To parameterize the WX algorithm, we maintain an interval $(r_1, r_2]$ and ensure $r^* \in (r_1, r_2]$ during the whole algorithm. Our algorithm follows the workflow of the WX algorithm. In each step of our algorithm, although we do not know r^*, we find some critical values of r in $(r_1, r_2]$ such that behaviors of the WX algorithm change on these critical values. Then the decision algorithm is called to do a binary search on these critical values and shrink the interval $(r_1, r_2]$ to $(r_1', r_2']$ such that no critical values lie in (r_1', r_2'). For any $r \in (r_1', r_2')$ after shrinking, if $r^* \neq r_2'$ (i.e., $r^* \in (r_1', r_2')$), then the behaviors of the WX algorithm running on r are the same as the behaviors of the WX algorithm running on r^*. The algorithm terminates after t is reached, and r^* is equal to r_2 of the final interval $(r_1, r_2]$. As t is reached within $O(n)$ steps and each step takes more than linear time (due to calling the decision algorithm), the total time of the algorithm is at least quadratic. To reduce the time complexity, we borrow an idea from [12] as follows. We classify each cell of the grid built in the WX algorithm as a *large cell* if it contains at least $n^{3/4} \log^{3/2} n$ points of P, and a *small cell* otherwise. We use a subroutine of the distance selection algorithm of Katz and Sharir [8] to preprocess all small cells and compute an interval $(r_1, r_2]$ such that if $r^* \neq r_2$, for any $r \in (r_1, r_2)$, the points lying in neighboring small cells have the same connectivities in both $G_r(P)$ and $G_{r^*}(P)$. More specifically, for any two points $p, q \in P$ lying in a pair of neighboring small cells of the grid, an edge connects p and q in $G_r(P)$ if and only if an edge connects p and q in $G_{r^*}(P)$. Then we follow the same algorithm as above. For small cells, we just pick any $r \in (r_1, r_2)$ and run the original WX algorithm on $G_r(P)$ since the points lying in neighboring small cells have the same connectivities in $G_r(P)$ and $G_{r^*}(P)$. This avoids parametric search (and thus calling the decision algorithm is not needed). For large cells, we run the same parametric search as before. The threshold $n^{3/4} \log^{3/2} n$ is carefully chosen to balance the running time of the parametric search on large cells (e.g., there are only $O(n^{1/4}/\log^{3/2} n)$ large cells in the grid) and the preprocessing on small cells. The total time complexity of our algorithm for the L_2 weighted RSP problem is $O(n^{5/4} \log^{5/2} n)$.

For the L_1 RSP problem, we use an approach similar to the distance selection algorithm of Katz and Sharir [8]. 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 [3, 4, 12] (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 [4]). 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 [8].

Remarks. Our RSP problem is defined with respect to a pair of points (s, t). Our techniques can be extended to solve a more general "single-source" version of the problem: Given a source point $s \in P$ and a value $\lambda > 0$, compute the smallest value r^* such that the shortest paths lengths from s to all vertices of $G_r(P)$ are at most λ, i.e., $\max_{t \in P} d_{r^*}(s, t) \leq \lambda$. The decision problem (i.e., deciding whether $r^* \leq r$ for a given r) now becomes deciding whether $\max_{t \in P} d_r(s, t) \leq \lambda$. All decision algorithms for our original RSP problem are actually for single-source-shortest-paths and thus can be used directly for solving the new decision problem with asymptotically the same time complexities. With these "new" decision algorithms, to compute r^*, we can follow the same algorithm schemes as before. One difference is that in the L_2 case our original algorithm terminates once t is reached but now we instead halt the algorithm once all points of P are reached, which does not affect the running time asymptotically. As such, the "single-source" version of the RSP problem in the L_2 weighted case can be solved in $O(n^{5/4} \log^{5/2} n)$ time and the L_1 unweighted/weighted case can be solved in $O(n \log^3 n)$ time.

References

[1] P.K. Agarwal, A. Efrat, and M. Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29:912–953, 1999.

[2] D. Burton and P.L. Toint. On an instance of the inverse shortest paths problem. *Mathematical Programming*, 53:45–61, 1992.

[3] S. Cabello and M. Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry: Theory and Applications*, 48(4):360–367, 2015.

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

[5] T.M. Chan and D. Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG)*, pages 24:1–24:13, 2018.

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

[7] H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. 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)*, pages 2495–2504, 2017.

[8] M. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing*, 26(5):1384–1408, 1997.

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

[10] H. Wang and J. Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. *Discrete and Computational Geometry*, 64:1141–1166, 2020.

[11] H. Wang and Y. Zhao. An optimal algorithm for L_1 shortest paths in unit-disk graphs. In *Proceedings of the 33rd Canadian Conference on Computational Geometry (CCCG)*, pages 211–218, 2021.

[12] H. Wang and Y. Zhao. Reverse shortest path problem for unit-disk graphs. In *Proceedings of the 17th International Symposium of Algorithms and Data Structures (WADS)*, pages 655–668, 2021. Full version available at https://arxiv.org/abs/2104.14476.

[13] J. Zhang and Y. Lin. Computation of the reverse shortest-path problem. *Journal of Global Optimization*, 25(3):243–261, 2003.