Computational Geometry 48 (2015) 360–367

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Shortest paths in intersection graphs of unit disks

Sergio Cabello [a,b,*,1], Miha Jejčič [c]

[a] Department of Mathematics, IMFM, Slovenia
[b] Department of Mathematics, FMF, University of Ljubljana, Slovenia
[c] Faculty of Mathematics and Physics, University of Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 19 February 2014
Accepted 3 December 2014
Available online 10 December 2014

Keywords:
Shortest path
Intersection graph
Unit-disk graph
Geometric graph

ABSTRACT

Let G be a unit disk graph in the plane defined by n disks whose positions are known. For the case when G is unweighted, we give a simple algorithm to compute a shortest path tree from a given source in $\mathcal{O}(n \log n)$ time. For the case when G is weighted, we show that a shortest path tree from a given source can be computed in $\mathcal{O}(n^{1+\varepsilon})$ time, improving the previous best time bound of $\mathcal{O}(n^{4/3+\varepsilon})$.

1. Introduction

Each set $\mathcal{S}$ of geometric objects in the plane defines its intersection graph in a natural way: the vertex set is $\mathcal{S}$ and there is an edge ss' in the graph, $s, s' \in S$, whenever $s \cap s' \neq \emptyset$. It is natural to seek faster algorithms when the input is constraint to geometric intersection graphs. Here we are interested in computing shortest path distances in unit disk graphs, that is, the intersection graph of equal sized disks.

A unit disk graph is uniquely defined by the centers of the disks. Thus, we will drop the use of disks and just refer to the graph $G(P)$ defined by a set P of n points in the plane. The vertex set of $G(P)$ is P. Each edge of $G(P)$ connects points p and p' from P whenever $\|p - p'\| \leq 1$, where $\|\cdot\|$ denotes the Euclidean norm. See Fig. 1 for an example of such a graph. Up to a scaling factor, $G(P)$ is isomorphic to a unit disk graph. In the *unweighted* case, each edge $pp' \in E(G(P))$ has unit weight, while in the *weighted* case, the weight of each edge $pp' \in E(G(P))$ is $\|p - p'\|$. In all our algorithms we assume that P is known. Thus, the input is P, as opposed to the abstract graph $G(P)$.

Exact computation of shortest paths in unit disks is considered by Roditty and Segal [15], under the name of *bounded leg* shortest path problem. They show that, for the weighted case, a shortest path tree can be computed in $\mathcal{O}(n^{4/3+\varepsilon})$ time. They also note that the dynamic data structure for nearest neighbors of Chan [6] imply that, in the unweighted case, shortest paths can be computed in $\mathcal{O}(n \log^6 n)$ expected time. (Roditty and Segal [15] also consider data structures to $(1+\varepsilon)$-approximate shortest path distances in the intersection graph of congruent disks when the size of the disks is given at query time; they improve previous bounds of Bose et al. [4]. In this paper we do not consider that problem.)

* Corresponding author.
E-mail addresses: sergio.cabello@fmf.uni-lj.si (S. Cabello), jejcicm@gmail.com (M. Jejčič).

[1] Supported by the Slovenian Research Agency, program P1-0297, projects J1-4106 and L7-5459, and by the ESF EuroGIGA project (project GReGAS) of the European Science Foundation.

http://dx.doi.org/10.1016/j.comgeo.2014.12.003
0925-7721/

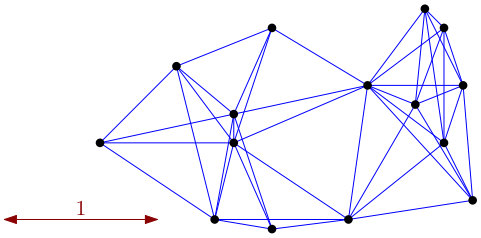

Fig. 1. Example of graph $G(P)$.

Alon Efrat pointed out that a semi-dynamic data structure described by Efrat, Itai and Katz [9] can be used to compute in $\mathcal{O}(n \log n)$ time a shortest path tree in the unweighted case. Given a set of n unit disks in the plane, they construct in $\mathcal{O}(n \log n)$ time a data structure that, in $\mathcal{O}(\log n)$ amortized time, finds a disk containing a query point and deletes it from the set. By repetitively querying this data structure, one can build a shortest path tree from any given source in $\mathcal{O}(n \log n)$ time in a straightforward way. At a very high level, the idea of the data structure is to consider a regular grid of constant-size cells and, for each cell of the grid, to maintain the set of disks that intersect it. This last problem, for each cell, reduces to the maintenance of a collection of upper envelopes of unit disks. Although the data structure is not very complicated, programming it would be quite challenging.

For the unweighted case, we provide a *simple* algorithm that in $\mathcal{O}(n \log n)$ time computes a shortest path tree in $G(P)$ from a given source. Our algorithm is implementable and considerably simpler than the data structure discussed in the previous paragraph or the algorithm of Roditty and Segal. For the weighted case, we show how to compute a shortest path tree in $\mathcal{O}(n^{1+\varepsilon})$ time. (Here, ε denotes an arbitrary positive constant that we can choose and affects the constants hidden in the $\mathcal{O}$-notation.) This is a significant improvement over the result of Roditty and Segal. In this case we use a simple modification of Dijkstra's algorithm combined with a data structure to dynamically maintain a bichromatic closest pair under an Euclidean weighted distance.

Gao and Zhang [12] showed that the metric induced by a unit disk graph admits a compact well separated pair decomposition, extending the celebrated result of Callahan and Kosaraju [5] for Euclidean spaces. For making use of the well separated pair decomposition, Gao and Zhang [12] obtain a $(1+\varepsilon)$-approximation to shortest path distance in unit disk graphs in $\mathcal{O}(n \log n)$ time. Here we provide exact computation within comparable bounds.

Chan and Efrat [7] consider a graph defined on a point set but with more general weights in the edges. Namely, it is assumed that there is a function $\ell: \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_+$ such that the edge pp' gets weight $\ell(p, p')$. Moreover, it is assumed that the function $\ell(p, p')$ is increasing with $\|p - p'\|$. When $\ell(p, p') = \|p - p'\|^2 f(\|p - p'\|)$ for a monotone increasing function f, then a shortest path can be computed in $\mathcal{O}(n \log n)$ time. Otherwise, if ℓ has constant size description, a shortest path can be computed in roughly $\mathcal{O}(n^{4/3})$ time.

There has been a vast amount of work on algorithmic problems for unit disk and a review is beyond our possibilities. In the seminal paper of Clark, Colbourn and Johnson [8] it was shown that several $\mathcal{NP}$-hard optimization problems remain hard for unit disk graphs, although they showed the notable exception that maximum clique is solvable in polynomial time. Hochbaum and Maass [13] gave polynomial time approximation schemes for finding a largest independent set problems using the so-called shifting technique and there have been several developments since.

Shortest path trees can be computed for unit disk graphs in polynomial time. One can just construct $G(P)$ explicitly and run a standard algorithm for shortest paths. The main objective here is to obtain a faster algorithm that avoids the explicit construction of $G(P)$ and exploits the geometry of P. There are several problems that can be solved in polynomial time, but faster algorithms are known for geometric settings. A classical example is the computation of the minimum spanning tree of a set of points in the Euclidean plane. Using the Delaunay triangulation, the number of relevant edges is reduced from quadratic to linear. For more advanced examples see Vaidya [16], Efrat, Itai and Katz [9], Eppstein [11], or Agarwal, Overmars and Sharir [3].

Organization In Section 2 we consider the unweighted case and in Section 3 we consider the weighted case. We conclude listing some open problems.

2. Unweighted shortest paths

In this section we consider the unweighted version of $G(P)$ and compute a shortest path tree from a given point $s \in P$. Pseudocode for the eventual algorithm is provided in Fig. 2. Before moving into the details, we provide the main ideas employed in the algorithm.

As it is usually done for shortest path algorithms we use tables dist$[\cdot]$ and $\pi[\cdot]$ indexed by the points of P to record, for each point $p \in P$, the distance $d(s, p)$ and the ancestor of p in a shortest (s, p)-path. We start by computing the Delaunay triangulation $DT(P)$ of P. We then proceed in rounds for increasing values of i, where at round i we find the set W_i of points at distance exactly i in $G(P)$ from the source s. We start with $W_0 = \{s\}$. At round i, we use $DT(P)$ to grow a neighborhood around the points of W_{i-1} that contains W_i. More precisely, we consider the points adjacent