A Linear Time Euclidean Spanner On Imprecise Points

Jiemin Zeng Jie Gao
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, USA
{jiezeng, jgao}@cs.stonybrook.edu

Abstract

An s-spanner on a set of points S in $\mathbb{R}^d$ is a graph on S where for every two points $p, q \in S$, there exists a path between them in G that is less than or equal to $s \cdot |pq|$ where $|pq|$ is the Euclidean distance between p and q. In this paper we consider the construction of an Euclidean spanner for imprecise points. In particular, we are given in the first phase a set of circles with radius r as the imprecise positions of a set of points and we preprocess them in $O(n \log n)$ time. In the second phase, the accurate positions of the points are revealed, one point for every circle, and we construct the $(1+\varepsilon)$-spanner in time $O(n \cdot (r+\frac{1}{\varepsilon})^{2d} \cdot \log(r+\frac{1}{\varepsilon}))$. The second phase can also be considered as an algorithm to quickly update the spanner. Our algorithm does not have any restrictions on the distribution of the points. It is the first such algorithm with linear running time.

1 Introduction

While in classical computational geometry all input values are assumed to be accurate, in the real world this assumption does not always hold. This imprecision can be modeled in many ways and they vary based on their applications [6, 8, 10, 11]. A popular model assumes that before the precise input points are known, we know the region (lines [4], circles/balls [1,3,7,9], fat regions [2, 12], etc) where each point lies in.

We aim to compute a $(1+\varepsilon)$-spanner on a set of imprecise points. In the model our algorithm operates in, an imprecise point p is defined to be a disk centered at a point $\dot{p}$ with radius r (which is dependent on the distance between the closest pair of points). The assumption is that initially, the only information of an input point is p and that later, the precise location of the point $\hat{p}$ (located somewhere within p) is revealed. Given a set of n points, an Euclidean spanner defines a graph G on the points, each edge weighted by the Euclidean length, such that the shortest path between any two points u, v in the graph is at most $1+\varepsilon$ times the Euclidean distance of u, v.

Our algorithm preprocess a set $S = \{p_1, p_2, \ldots, p_n\}$ of n imprecise points in $O(n \log n)$ time such that when $\hat{S} = \{\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_n\}$ is available, we can compute a $(1+\varepsilon)$-spanner in $O(n \cdot (r+\frac{1}{\varepsilon})^{2d} \cdot \log(r+\frac{1}{\varepsilon}))$ time where d is the dimension of the input. It is important to note that our algorithm will accept input sets with overlapping points of any depth.

2 Algorithm Definition

The algorithm we present constructs a $(1+\varepsilon)$-spanner or more specifically the deformable spanner (DefSpanner) as described in [5]. A DEFSPANNER is a specific $(1+\varepsilon)$-spanner construction that is designed to be easily modified and updated. More specifically, for a set of points S in $\mathbb{R}^d$, the DEFSPANNER is made up of a hierarchy of levels $S_{\lceil \log_2 \alpha \rceil} \subseteq S_{\lceil \log_2 \alpha \rceil - 1} \subseteq \cdots \subseteq S_i \subseteq S_{i-1} \subseteq \cdots \subseteq S_0 = S$ where $|S_{\lceil \log_2 \alpha \rceil}| = 1$. Any set S_i is a maximal subset of S_{i-1} where for any two points $p, q \in S_i$, $|pq| \geq 2^i$. Also, every node $p \in S_{i-1}$ is assigned a parent node q

in the level above where $|pq| \leq 2^i$. The edges of a DefSpanner G of a set S are determined by connecting nodes within distance $c \cdot 2^i$ in level i, where $c = 2r + \frac{16}{\varepsilon} + 4$. Such nodes are called neighbors.

In the preprocessing phase, a DefSpanner $\dot{G}$ is constructed with the point set $\dot{S}$ as in [5]. The running time is $O(n \log n)$.

When the true positions of the points are revealed, we construct a DefSpanner $\hat{G}$ for $\hat{S}$ by inserting the points one by one into $\hat{G}$. Initially, nodes are inserted in their top-down order in $\hat{G}$, but the order may change as the algorithm executes. We insert each node into the bottom level of $\hat{G}$ if other nodes in the same level have been inserted, otherwise a new level is added below. For each node $\hat{p}$ that we insert into level i, there are three phases in our algorithm.

Step 1: Check for demotions. First, we compare the distance between $\hat{p}$ and any of it's neighbors that have been already inserted into $\hat{G}$. If any of the distances are less than 2^i, then we demote the node to a lower level by delaying it's insertion.

Step 2: Find a parent. We compare the distances between $\hat{p}$ and it's old parent and subsequently the old parent's neighbors. If none of those nodes are a suitable parent, we promote $\hat{p}$ up the hierarchy and repeat our tests with it's old grandparent and the old grandparent's neighbors. We repeat and promote $\hat{p}$ until a parent is found or $\hat{p}$ resides at the top of $\hat{G}$. In certain cases, instead of testing the old parent, we compare distances with an ancestor first or with the node that was considered to be too close (if $\hat{p}$ was demoted).

Step 3: Find all neighbors. We compare distances with $\hat{p}$ and it's cousins ($\hat{p}$'s parent's neighbors' children) in every level it resides in. We also check to see if any of the nodes are too close (less than 2^i in level i). In this case (which only occurs with nodes that have been previously demoted in the first step), we demote the node again.

The algorithm terminates when all nodes have been inserted into $\hat{G}$. The cost of preprocessing is the DefSpanner construction cost or $O(n \log_2 \alpha(r + 1/\varepsilon)^d)$ [5]. The running time of our algorithm is $O(n \cdot (r + \frac{1}{\varepsilon})^{2d} \cdot \log(r + \frac{1}{\varepsilon}))$.

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures*, WADS '09, pages 1–12, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Preprocessing imprecise points for delaunay triangulation: Simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

[3] O. Devillers. Delaunay triangulation of imprecise points, preprocess and actually get a fast query time. *JoCG*, pages 30–45, 2011.

[4] E. Ezra and W. Mulzer. Convex hull of imprecise points in o(n log n) time after preprocessing. In *Proceedings of the 27th annual ACM symposium on Computational geometry*, SoCG '11, pages 11–20, New York, NY, USA, 2011. ACM.

[5] J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *In Proc. of the 20th ACM Symposium on Computational Geometry (SoCG04*, pages 179–199, 2004.

[6] L. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. In *Proceedings of the international symposium on Algorithms*, SIGAL '90, pages 261–270, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[7] M. Held and J. S. B. Mitchell. Triangulating input-constrained planar point sets. *Inf. Process. Lett.*, 109(1):54–56, Dec. 2008.

[8] M. Löffler and J. M. Phillips. Shape fitting on point sets with probability distributions. *CoRR*, abs/0812.2967, 2008.

[9] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom. Theory Appl.*, 43(3):234–242, Apr. 2010.

[10] T. Nagai, S. Yasutome, and N. Tokura. Convex hull problem with imprecise input. In *Revised Papers from the Japanese Conference on Discrete and Computational Geometry*, JCDCG '98, pages 207–219, London, UK, UK, 2000. Springer-Verlag.

[11] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *Proceedings of the fifth annual symposium on Computational geometry*, SCG '89, pages 208–217, New York, NY, USA, 1989. ACM.

[12] M. van Kreveld, M. Löffler, and J. S. B. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM J. Comput.*, 39(7):2990–3000, June 2010.