Experimentation Project on Geometric t-Spanners

Research report for Geometric Algorithms (2IL55) – Autumn semester 2014

Daan Creemers
Student number: 0748693
d.j.a.creemers@student.tue.nl

Pieter-Paul Kramer
Student number: 0675271
p.kramer@student.tue.nl

Qi Xiao
Student number: 0925490
xiaqqaix@gmail.com

January 8, 2015

Abstract

This paper presents an experimental study comparing the performance and quality of four geometric spanner algorithms: the greedy spanner, a spanner based on well-separated pair decomposition, the Yao graph and the Θ-graph. The algorithms are tested on different challenging sets of points in the Euclidean plane. The results are compared to the theoretical results. The considered measures include the number of edges, running time and number of intersections.

1 Introduction

Consider a set V of n points in $\mathbb{R}^d$, where the dimension d is constant. Set V defines the set of vertices of an undirected graph G. The weight between two vertices u and v is $wt(u, v)$. In $\mathbb{R}^2$ the weight of an edge (u, v) is the euclidean distance $|uv|$ between the two endpoints of the edge. Given $t > 1$ be a real number. A graph $G = (V, E)$ is a geometric t-spanner if for each pair of points $u, v \in V$ there exists a path using E and the length of the path is at most $t \cdot |uv|$. This path is called a t-path. Parameter t is called the stretch factor or dilation factor of the spanner.

In this paper we present an experimental study comparing the performance and quality of a couple of well-known spanner algorithms for points in the Euclidean plane. We consider the greedy spanner, a spanner based on well-separated pair decomposition, the Yao graph and the Θ-graph. The measures we use to evaluate the algorithms are the stretch factor, the number of edges in the spanner (also known as size), the sum of the length of the edges in the spanner (also known as weight), the maximum degree of any vertex in the spanner and the number of intersections. Besides these measures on the structure of spanners, we use the running time to compare the performance of the algorithms.

Creating a complete graph often represents the ideal situation in communication networks, for example distributed systems or routing in wireless networks. In both systems every vertex

must be able to communicate to every other vertex in the graph. Using a complete graph guarantees a connection between each pair of vertices, but maintaining a complete graph is expensive when considering edges for example as wires between base stations. In contrast to this complete network, a sparse network might also be able to fulfill all specifications by requiring that the network is a t-spanner for a reasonable t. Instead of using a direct link between two vertices, a message now has to traverse a path through the graph. The traversed distance is at most t times the original distance. Other applications for geometric spanners include motion planning, network topology design and transportation network construction.

Work on spanners in wireless networks can be found in a number of papers. For a recent article see, for example, Abu-Affash et al. [2] where they introduce a spanner to determine the transmission power to each of the nodes of a wireless network such that the cost of the resulting spanner is low. Zhu et al. [10], Shpungin and Segal [8] and Molla [6] also conducted research on spanner construction in wireless networks which shows that this is still a interesting area. Another possible research direction is kinetic spanners. Abam and De Berg [1] studied the problem of efficiently maintaining spanners while the vertices move. Their $(1 + \varepsilon)$-spanner can handle events in time $O(n^2/\varepsilon^{d-1})$ assuming that the trajectories of the vertices can be described by bounded-degree polynomials.

The remainder of this paper is organized as follows. First the algorithms that are implemented and why they are chosen are discussed. Next section 3 describes the measures that are used during testing. In section 4 we consider the datasets used for testing and argue why these datasets are suitable for the purpose of this study. Section 5 reviews the implementation of the algorithms. In section 6 the experiment results are discussed. Finally, section 7 concludes the paper with the findings.

2 Algorithms

The following algorithms have been implemented and tested: the greedy spanner, the WSPD spanner, the Yao graph and the Θ-graph. We now introduce these algorithms briefly.

2.1 Greedy Spanner

The greedy spanner algorithm [3], like most greedy algorithms, work in an incremental manner. Starting from a graph with no edges, it adds the shortest possible edge uv such that the total weight of the path between uv is greater than $t|uv|$. By repeating this step until no edges can be added, the dilation ratio becomes lower than t. Intuitively, the edge added at each step is the most "reasonable" edge to add; indeed, as the experimental results show, the properties of the greedy spanner are pretty good in most cases.

A known, optimal implementation of the greedy spanner runs in $O(n \log n)$ time [7].

2.2 WSPD Spanner

The WSPD spanner [4] works by first finding a s-well-separated point decomposition of the vertices. After that it simply adds an edge between each pair of vertex sets in the WSPD; when a vertex set contains more than one vertices, an arbitrary one can be chosen. The value of s can be derived from the desired dilation ratio by the formula $s = 4(t + 1)/(t - 1)$. See Figure 1 for an example with 8 vertices where the complete graph is much larger the WSPD-spanner.

The running time of the WSPD spanner algorithm depends primarily on that of finding WSPD, which can be done in $O(n \log n)$ time.

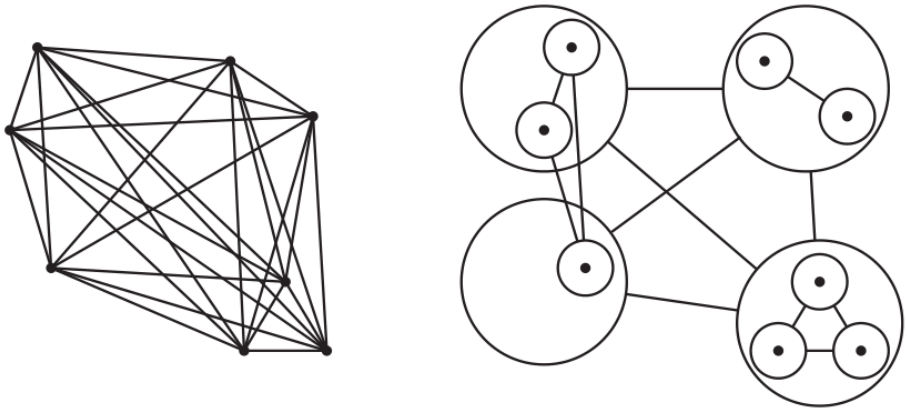

Figure 1: A complete graph compared to the well-separated pair decomposition. (Source: lecture 'Geometric Spanners and Well Separated Pair Decompositions' from the course Geometric Algorithms at University of Technology Eindhoven)

2.3 Yao graph and Θ-graph

The Yao graph [9] and Θ-graph [5] algorithms are *cone-based* spanner algorithms; for each vertex v, they subdivide the space to k non-overlapping cones (each of angle $2\pi/k$) and connect to a certain point in each cone.

In the Yao graph algorithm, the nearest neighbors in each cone is connected to v. The Θ-graph is slightly more sophisticated; it connects the vertex whose projection on the cone bisector is closest to v to v. This approach, though seemingly more unnatural, lends itself to an efficient scanning algorithm that runs in $O(n \log n)$ time.

The Yao graph and Θ-graph have similar properties. The dilation factor of the Yao graph is bounded by $1/(\cos\theta - \sin\theta)$ (where $\theta = 2\pi/k$). The same bound applies to the Θ-graph when $k \geq 9$.

3 Measures

In the experimentation we test the following measures of the resulting spanners:

1. Size (number of edges) and total weight. These are the most straightforward measures on the overall complexity of the spanner. In many applications (e.g. traffic networks) they can be used to estimate the total construction cost of the network.

2. (Actual) stretch factor (dilation ratio). Since the t we are given is really only an upper bound on the stretch factor, it is possible that an algorithm will yield a spanner with a stretch factor much lower than t. By computing the actual value we can know how "tight" the algorithm is in terms of the stretch factor.

3. Maximum degree of vertices. This measure reflects the local complexity of the spanner; a low maximum degree often reflects high resilience of the network. For instance, in communication networks, the total throughput and bandwidth at a given vertex is often bounded, and it is often desirable to have spanners where the edges are more spread apart instead of concentrating on a few vertices.

4. Number of intersections. This measure is another aspect on the overall complexity of the spanner. For instance, if the spanner is to be visualized, a smaller number of intersections leads to better readability. In communication networks, a smaller number of intersections means less signal interference.

5. Running time. We would like to analyze the running time of the algorithms and compare them to theoretical results.

4 Data sets

We have run our experiments on different types of data sets to determine the performance and the quality of the spanner algorithms. For each of the data sets, we use the dilation ratios $(1.3, 1.5, 1.75, 2, 2.5, 3, 4, 5)$.

1. Uniformly distributed points. We test on data sets of 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350 points uniformly distributed in the area $[0, 100] \times [0, 100]$.

2. Data sets from the 2013-2014 data challenge of the course Geometric Algorithms at University of Technology Eindhoven. The data challenge includes six data sets which are summarized in Table 1. These data sets are created by student of the course of last year. As can be seen from the table, there is a great diversity in the number of vertices. Using large data sets can show the limits of the algorithms if the running time exceptionally increases. The range of the X-axis and Y-axis also differ considerably. This is done to make sure that algorithms that scale the input also work for larger values.

3. Real-world dataset. We created a mixed data set containing all train stations in The Netherlands as well as all bus stops in Eindhoven. Because we could not obtain a data set containing all bus stops, these are not the actual bus stops but rather random points in the region of Eindhoven. We scaled the 442 GPS coordinates to integers by multiplying them by 10^4. The idea of the data set is to test how well the algorithm deals with points on different scales, which might break implementations that use normalizing length or that assume in some way that close pairs are distributed over the map. Additionally, the map contains real and realistic data which is also interesting in its own right.

5 Implementation

To perform experiments the algorithms were implemented in Python 2.7.3. Python is an interpreted programming language that allows for rapid development. To efficiently perform computations with large amounts of numerical data (in terms of speed and storage) NumPy 1.6.1 was used. The experiments were performed on a HP 8530w notebook with an Intel Centrino Core 2 Duo 2.8GHz processor and 4GB of RAM memory.

Data set	Number of vertices	Dilation ratio	X_{Min}	X_{Max}	Y_{Min}	Y_{Max}
B1	8	2.00	0	14	0	3
B2	2,689	2.00	1,498.728	1,501,272	0	75
B3	10,001	1.42	1	5,001	1	5,001
B4	1,000	1.10	0	1,330	0	1,333
B5	42	1.25	54	520	37	911
B6	10,000	1.10	85,000	214,999	100,000	199,999

Table 1: Data sets from the data challenge

The algorithms were not optimized to achieve an optimal (asymptotical) running time, mainly because that is outside the scope of this study. As a consequence, the asymptotical running time of the greedy algorithm is $O(n^3 \log n)$, and that of the Yao- and Θ-graph algorithms is $O(n^2 \log n)$ where n is the number of vertices on which the spanner is to be computed.

The algorithms were then combined in an experimentation environment which computes a spanner for each input graph using each of the algorithms with each of the chosen dilation ratios . It then computes the relevant measures and stores these for later analysis.

5.1 Greedy Spanner

The greedy algorithm was implemented in the most basic form: first the distances between each node pair are computed and sorted in $O(n^2 \log n)$ time, and then the algorithm successively checks each node pair to see if the dilation ratio between that node pair is sufficiently low. For this computation the length of the shortest path on the graph is computed using Dijkstra's algorithm, in $O(n \log n)$ time for each of the n^2 pairs, yielding a total running time $O(n^3 \log n)$.

5.2 WSPD

The implementation of the WSPD-spanner is not yet finished.

5.3 Yao graph and Θ-graph

We implement a naïve algorithm for finding the Yao graph. For each vertex v, we iterate each vertex u other than v and find the cone u is in and the distance between u. An edge is created between v and the nearest vertex in each cone along the way. The total running time is $O(n^2)$.

The Θ-graph algorithm is implemented in a similar way, but instead of finding the distances from u to v, we find the distance from u to the projection of v on the cone bisector. The running time is also $O(n^2)$.

6 Experiment Results

The experiments have been performed as described in the previous sections. This section describes and discusses the results that were obtained.

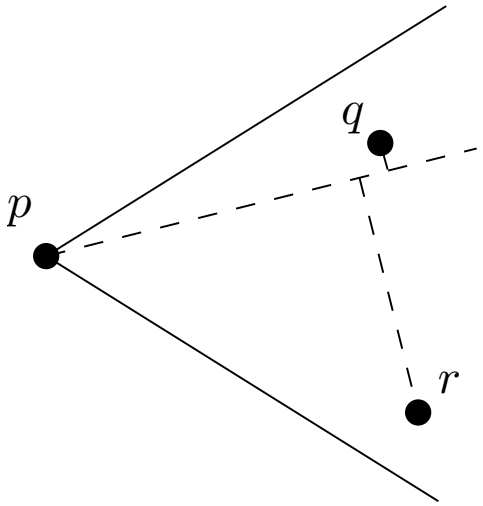

Figure 2: Point p in the cone is connected to q in the Yao graph while it is connected to r in the Theta graph.

Because of the rapid increase of the running time with the number of vertices in the input set, a number of experiments were computationally unfeasible. The greedy algorithm was not executed for input graphs of more than 450 vertices and the Yao- and Θ-graph algorithms were not executed for input graphs of more than 3000 vertices.

6.1 Running time

As expected from the asymptotic running times, the greedy algorithm is much slower than the Yao and Θ-graph algorithms, which perform nearly identically. This is depicted in Figure 3. It can also be seen that the Θ-graph algorithm is slightly faster than the Yao algorithm, which is probably due to the fact that computing the distance along the bisector is computationally cheaper than computing the Euclidean distance.

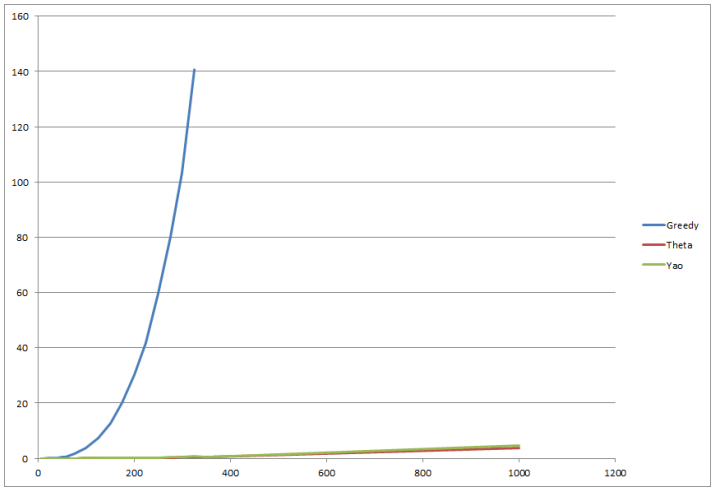

Figure 3: Running times of Greedy, Yao- and Θ-graph algorithms on input graphs of various sizes.

6.2 Number of edges

The number of edges used by the algorithms form an almost perfectly linear relation with the number of vertices in the input set, as is depicted in Figure 4. The Yao and Θ-graphs always have an identical number of edges because of how they introduce new edges (refer to section 2). In this example, which shows all spanners with maximum dilation ratio 5, the greedy spanner uses about 1.18 times as many edges as there are vertices in the input graph and the Yao and Θ-graph spanners use about 5.55 times as many edges as there are vertices in the input graph.

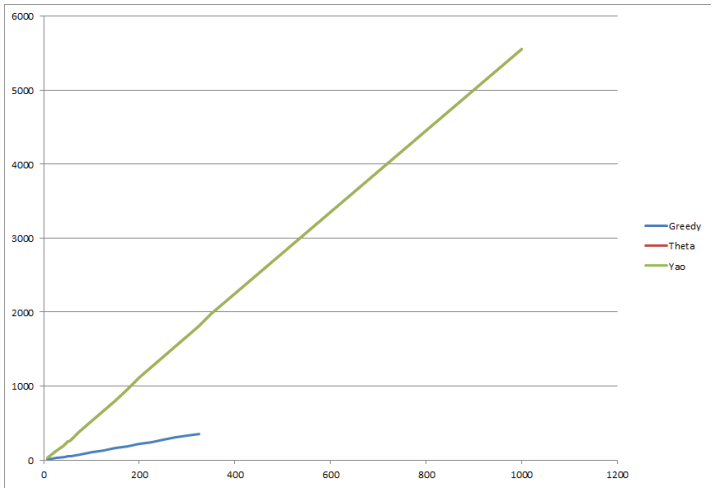

Figure 4: Number of edges of spanners generated by Greedy, Theta and Yao algorithms on input graphs of various sizes, all with maximum dilation ratio 5.

This is inherent to the way the Yao and Θ-graph spanners are constructed, as is discussed in subsection 6.3.

6.3 Actual dilation ratio

The maximum allowable dilation ratio is a parameter to the algorithms. However that does not mean the algorithms produce a spanner with exactly that dilation ratio. Usually the dilation ratio will be slightly smaller, sometimes even much smaller. This is in itself not a problem, a smaller dilation ratio can in fact be considered better better than a larger one, but it is also unnecessary so it should not come at the expense of other measures, such as the amount of edges.

Figures 5 and 6 show the actual dilation ratio of all constructed spanners versus the maximum allowable dilation ratio that was provided to the algorithm as a parameter. The greedy algorithm produces spanners that have a dilation ratio close to the maximum allowable dilation ratio in all cases. The Yao algorithm, on the other hand, produces only spanners with a dilation ratio smaller than 2. The Θ-algorithm has similar results to the Yao-algorithm.

Figures 7 and 8 show spanners constructed by the greedy algorithm and by the Yao algorithm. The Yao spanner has many more edges than the greedy spanner (as is discussed in section 6.2). But what is also very well visible here is that each vertex in the Yao spanner must have a neighbor 'in each direction', so to speak. This means that every vertex in the Yao spanner must have a relatively high degree. This is opposed to the greedy spanner, where

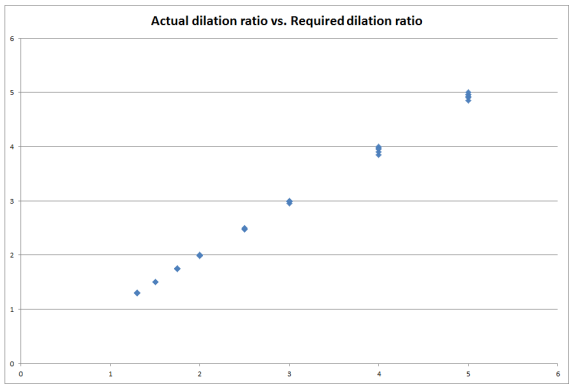

Figure 5: Actual dilation ratios of spanners generated by the greedy algorithm for each given maximum allowable dilation ratio.

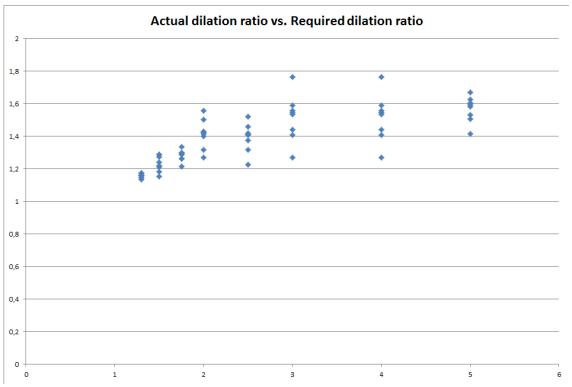

Figure 6: Actual dilation ratios of spanners generated by the greedy algorithm for each given maximum allowable dilation ratio. Note that the scaling differs from that in figure 5.

many vertices are on paths connecting vertices that are close to each other. These vertices have a degree of only two or even one for the endpoints of these paths. This observation explains why Yao spanners have a relatively high amount of edges and also a low dilation ratio, even when a low dilation ratio is not required.

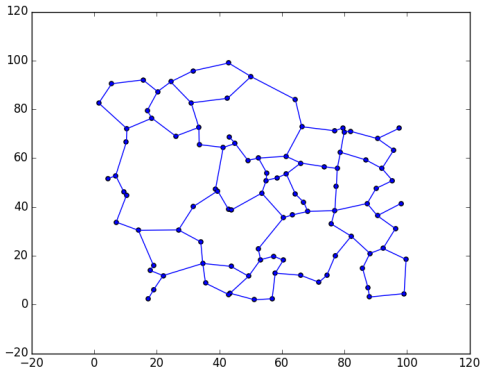

Figure 7: Greedy spanner with $t = 3$.

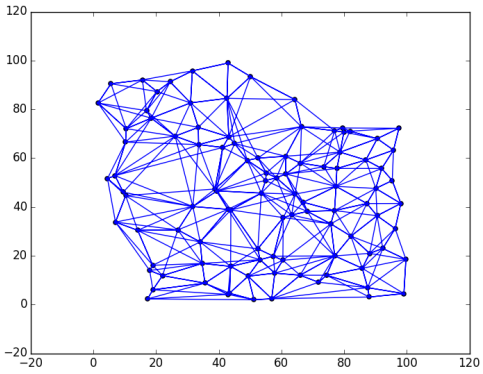

Figure 8: Yao graph with $t = 3$.

7 Conclusion

In this paper we have compared experimental results of four geometric spanner algorithms. The algorithms that were implemented are the greedy spanner, the Yao graph, the Θ-graph and a spanner based on well-separated point decomposition. We have considered the size, actual stretch factor and running time. We tested on randomly distributed data sets, a real-world data set with trains stations in the Netherlands and Luxembourg and six data sets from a data challenge for geometric spanners.

The experiments show that the greedy spanner has considerably less edges than both the Yao and Θ-graph. A result of the high number of edges is that the vertices in the Yao and Θ-spanners have a relatively high degree compared to the greedy spanner where many edges are on a path. The disadvantage of the good quality of the greedy spanner is the high running time. While the Yao and Θ-graph have a running time of $O(n^2)$, the greedy spanner needs $O(n^3 \log n)$. We furthermore found that the greedy spanner has a dilation ratio close to the maximum allowed dilation ratio while the Yao and Θ-algorithms only produce spanners with dilation ratio smaller than 2, a consequence of the Yao and Θ-algorithms adding more edges than necessary.

References

[1] Mohammad Ali Abam and Mark de Berg. Kinetic spanners in $\mathbb{R}^d$. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 43–50. ACM, 2009.

[2] A Karim Abu-Affash, Rom Aschner, Paz Carmi, and Matthew J Katz. Minimum power energy spanners in wireless ad hoc networks. *Wireless Networks*, 17(5):1251–1258, 2011.

[3] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[4] Paul B Callahan and S Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[5] Ken Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 56–65. ACM, 1987.

[6] Nawar M El Molla. *Yao spanners for wireless ad hoc networks*. PhD thesis, Villanova University, 2009.

[7] Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. In *Algorithm Theory-SWAT 2000*, pages 314–328. Springer, 2000.

[8] Hanan Shpungin and Michael Segal. Near-optimal multicriteria spanner constructions in wireless ad hoc networks. *IEEE/ACM Transactions on Networking (TON)*, 18(6):1963–1976, 2010.

[9] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

[10] Ying Zhu, Minsu Huang, Siyuan Chen, and Yu Wang. Cooperative energy spanners: energy-efficient topology control in cooperative ad hoc networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 231–235. IEEE, 2011.