Visualization of Geometric Spanner Algorithms

Mohammad Farshi[1] **and Seyed Hossein Hosseini**[2]

1 **Combinatorial & Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran**
`mfarshi@yazd.ac.ir`

2 **Combinatorial & Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran**
`hoseini.seyedhosein@gmail.com`

Abstract

It is easier to understand an algorithm when it can be seen in interactive mode. The current study implemented four algorithms to construct geometric spanners; the path-greedy, gap-greedy, Θ-graph and Yao-graph algorithms. The data structure visualization framework (`http://www.cs.usfca.edu/~galles/visualization/`) developed by David Galles was used. Two features were added to allow its use in spanner algorithm visualization: support point-based algorithms and export of the output to Ipe drawing software format. The interactive animations in the framework make steps of visualization beautiful and media controls are available to manage the animations. Visualization does not require extensions to be installed on the web browser. It is available at `http://cs.yazd.ac.ir/cgalg/AlgsVis/`.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases geometric spanner networks, geometric spanner algorithms animations.

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.67

Category Multimedia Contribution

1 Introduction

Algorithm visualization is a wonderful tool for educational and research purposes. It helps teachers transfer key ideas and the structure of even complex algorithms in a short period of time. In research, providing examples of the output of an algorithm for a specific input can encourage deeper understanding of the algorithm and allow use of the output in presentations.

A geometric t-spanner ($t \geq 1$) for a set of points $S \subset \mathbb{R}^d$ is an undirected Euclidean graph $G = (S, E)$ such that, for any two points $p, q \in S$, there is a path in G between p and q with a length of at most $t|pq|$ where $|pq|$ is the Euclidean distance between p and q. The length of a path is the sum of all edge lengths on the path.

Several algorithms, when given point set S and $t \geq 1$ as input, compute a t-spanner on S, see [7]. The applications offered by geometric spanners means that they are commonly-used to solve problems in computational geometry as well as in fields such as bio-informatics [8].

To the best of our knowledge, there are two tools to visualize some geometric spanner algorithms:

- The applet by Specht and Smid [9] makes the Θ-graph and the geometric neighborhood graph for a given point set. This applet mainly shows the t-path and the shortest path between points in a graph.

- The applet by Aschner [3] visualizes three spanner algorithms: path-greedy, Θ-graph, and Yao-graph algorithms.[1]

The visualization presented herein consists of four spanner algorithms: path-greedy, gap-greedy, Θ-graph and Yao-graph (see [5]). This visualization had the following specifications:

- The visualization animates the steps of the algorithms. The speed of animation is adjustable.
- The visualization is browser-independent and it does not require extension to be installed on a web browser. One can use it on any modern browser, including iOS devices like the iPhone and iPad, android devices such as smart watches and TVs, and even the web browser on Kindle.
- It can export the network generated by the algorithms to .ipe format. One can open and edit the output in Ipe drawing editor, commonly-used software that is well-known in the computational geometry community.

2 Algorithms

This section briefly explains the visualized algorithms.

2.1 Path-greedy algorithm

This algorithm uses a set of points and $t \geq 1$ as input and constructs a t-spanner of the input point set. It begins with a graph with the input point set as its vertex set and an empty edge set. In the first step, it sorts all pairs of points in non-descending order of distance. It then checks all pairs of points in sorted order. For each pair (u, v) of points, if the length of the shortest path between u and v in the current graph is greater than $t \times |uv|$, then (u, v) is added to the edge set of the graph. For more details one can see [1, 7].

2.2 Θ-graph and Yao-graph algorithms

These algorithms use a point set and $t \geq 1$ (or, equivalently, integer k) as input. For any point u in the input point set, the plane is divided into k non-overlapping cones with an apex at u. The number of cones, or k, is the smallest integer such that $t \leq 1/(\cos\theta - \sin\theta)$ for $\theta = 2\pi/k$. The Yao-graph algorithm connects u to the point nearest to u in each cone. The Θ-graph algorithm, for each cone, considers a fixed line through u in the cone. It projects all the points in the cone to this line and connects u to the point for which projection to the line through u is closest to u. For more details about the Θ-graph and the Yao-graph algorithms see [4, 10].

2.3 Gap-greedy algorithm

This algorithm uses a point set and two real numbers θ and w as input. The value of θ and w must be chosen such that $0 < \theta < \pi/4$ and $0 \leq w \leq (\cos\theta - \sin\theta)/2$ where t is $1/(\cos\theta - \sin\theta - 2w)$. The gap-greedy algorithm, as for the path-greedy algorithm, sorts all ordered pairs of points non-descending order and processes them in this order. It begins with a graph with the input point set as its vertices and an empty edge set. For each pair (u, v), if there is an edge in the current graph such that the angle between (u, v) and the edge is less than θ

[1] We could not run the applet to see how it works.

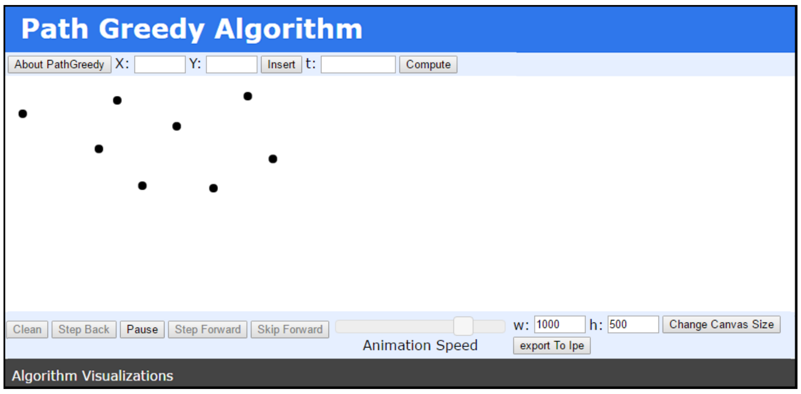

Figure 1 Screen before clicking *compute* button.

and u is close to the source of the edge or v is close to the sink of the edge, then it does not add the edge to the graph. For more details on the gap-greedy algorithm see [2, 7].

3 Usage and Implementation

This section, provides a brief description of the use of visualization and its properties and then describes the implementation of visualization.

3.1 How visualization is used

The visualization of an algorithm is based on interactive animation of the steps of the algorithm. The visualization screen includes two main control parts (Figures 1 and 2).

Algorithm controls: The user can add a point by left-clicking on the position of the point on the canvas or by inserting the coordinates of the point at the top of the page in front of X: and Y: and then clicking on the "insert" button. After adding all points, the user should enter t, k, θ, or w based on the algorithm and afterward click on the "compute" button. The steps for construction of the spanner using the algorithm appear one-by-one. One can remove any previously-inserted point by right-clicking on that point.

Animation controls: Animation controls appear at the bottom of the screen with which to manage animation using the following media controls:
- **clean:** reset the input to an empty canvas;
- **play/pause:** play and pause the animation;
- **step back/forward**: go one step backward/forward in the animation;
- **skip forward:** go to the final step of the algorithm.

A slider control is provided with which to manage animation speed and a button by which to export the output to Ipe drawing software. The user can change the size of the main screen (width and height) by changing the values and clicking on the "Change Canvas Size" button. Animation controls are the same for all algorithms.

3.2 Implementation details

Visualization uses the "structure visualization framework" developed by David Galles [6]. Any algorithm visualization set includes an HTML file and a javascript file. The algorithm

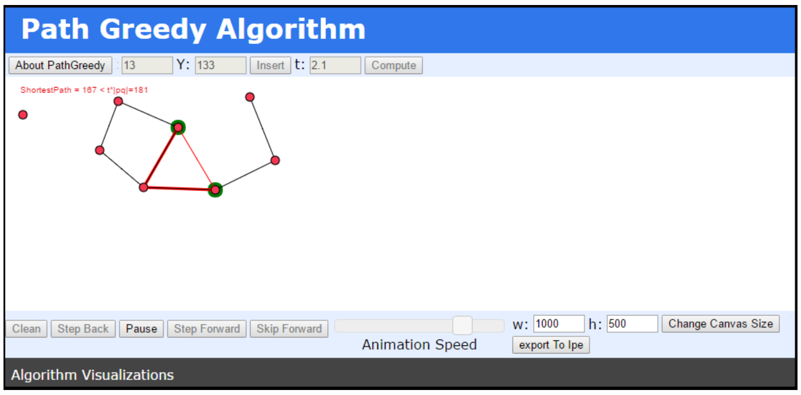

Figure 2 Screen after clicking *compute* button.

is coded in the javascript file. The body of programming is based on the canvas tag in html5 that is manipulated using javascript instructions. The canvas tag can be used to draw a shape and for animation via javascript. Paths, boxes, circles, and texts can be drawn in canvas and images can be added.

Two features have been added to the framework to allow its use in spanner algorithm visualization: support of point-based algorithms and export of the output to Ipe Drawing software format. The first is used to obtain the input point set from the user and the second to export the result of the algorithm to the input point set as an .ipe file.

Acknowledgements. The authors would like to thank Davood Bakhshesh for discussions on both the algorithms and their implementation.

References

1 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.
2 Sunil Arya and Michiel Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33–54, 1997.
3 Rom Aschner. Geometric spanners applet. `http://www.cs.bgu.ac.il/~romas/greedy.html`.
4 Ken Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. of the 19t Annual ACM Symp. on Theory of Computing*, pages 56–65. ACM, 1987.
5 Mohammad Farshi and Seyed Hossein Hosseini. Geometric spanner algorithms visualizations. `http://cs.yazd.ac.ir/cgalg/AlgsVis/`.
6 David Galles. Data structure visualizations. `http://www.cs.usfca.edu/~galles/visualization/`.
7 Giri Narasimhan and Michiel Smid. *Geometric spanner networks.* Cambridge University Press, 2007.
8 Daniel Russel and Leonidas J. Guibas. Exploring protein folding trajectories using geometric spanners. *Pacific Symposium on Biocomputing*, pages 40–51, 2005.
9 Petra Specht and Michiel Smid. Visualization of spanners. `http://isgwww.cs.uni-magdeburg.de/tspanner/spanner.html`.
10 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.