

# Browsing large graphs with MSAGLJS, a graph dragh drawing tool in JavaScript

Lev Nachmanson and Xiaoji Chen

Microsoft Research, US,  
levnach@hotmail.com, cxiaoji@gmail.com,  
Msagljs github home page: <https://github.com/microsoft/msagljs>

**Abstract.** There has been progress in visualization of large graphs recently. Still, interacting with a large graph in the browser with the same ease as browsing an online map, inspecting the high level structure and zooming to the lower details, is still an unsolved problem, in our opinion. In this paper we describe MSAGLJS's approach to two aspects of this problem. Firstly, we give a novel algorithm for edge routing, where the edges do not overlap the nodes. The algorithm does not necessarily creates optimal paths but is efficient and creates visually appealing paths. Secondly, to facilitate graph vizualization with DeckGL, namely fast zoom, and pan operations, and keeping the number of entities on the screen below a specific bound all the time, we use tiling. Our tiling procedure is simple and efficient. It is the second contribution of the paper.

## 1 Introduction

## 2 Related work

3 Links to large graph visualization

4 [1]

5 [2]

6 [3]

7 [4]

8 [5]

9 [6]

10 machine learing approach [7]

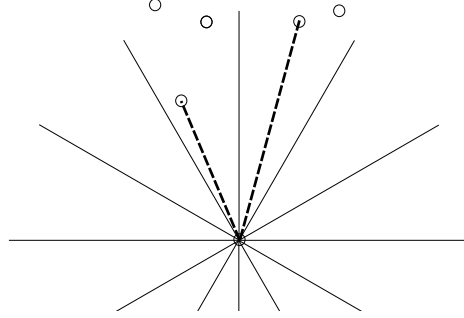
11 [8]

12 [9]

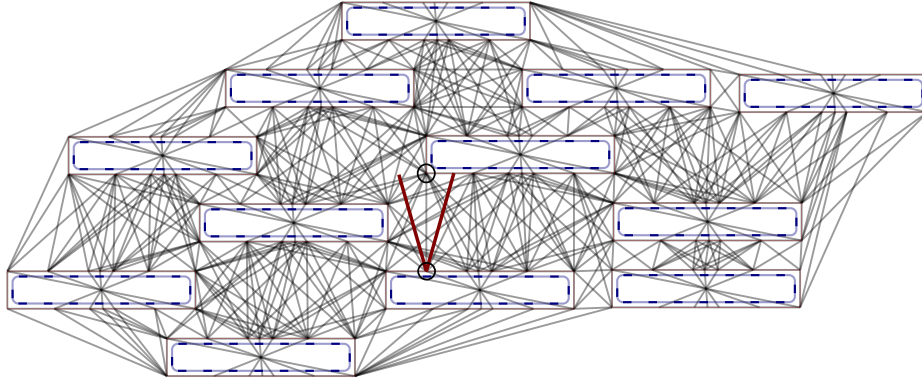
## 13 Edge routing

14 The edge routing starts, as in [10], by building a spanner graph, an approximation  
15 of the full visibility graph. The spanner, see Fig. 2, is built on a variation of a

16 Yao graph, which was introduced independently by Flinchbaugh and Jones [11]  
 17 and Yao [12]. A Yao graph is defined by the set of cones with the apices at  
 18 the vertices. The cones have the same angle, usually in the form of  $\frac{2\pi}{n}$ , where  
 19  $n$  is a natural number. This way the cones with the apex at a specific vertex  
 20 partition the plane as illustrated in Fig. 1. For each cone at most one edge is  
 21 created connecting the cone apex with a vertex inside of the cone.

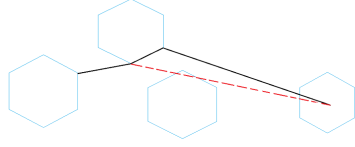


22 **Fig. 1.** Yao graph

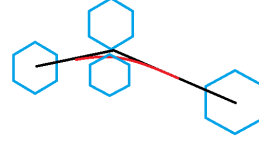


23 **Fig. 2.** Spanner graph is built using the idea of Yao graphs. The dashed curves are the  
 24 original node boundaries. Each original curve is surrounded by a polygon with some  
 25 offset to allow the polyline paths smoothing without intersecting the former.  
 26 The edge marked by the circles is created because the top vertex is inside of the cone  
 27 and it is the closest among such vertices to the cone apex. The apex of the cone is the  
 28 lower vertex of the edge.  
 29 MSAGLJS uses cone angle  $\frac{\pi}{6}$ , so the edges of the spanner can deviate from the optimal  
 30 direction by this angle. Therefore the shortest paths on the spanner have length that  
 31 is at most the optimal shortest length multiplied by  $\frac{1}{\cos(\frac{\pi}{6})} \simeq 1.155$ .

35 The approach of [10] first builds a polyline path through the spanner, and  
 36 then applies some local modifications to shorten and smoothen the path. For  
 37 shortening it tries to shortcut a vertex, as illustrated in Fig 3. To smoothen it



**Fig. 3.** Unsuccessful shortcut



**Fig. 4.** Fitting a Bezier segment into a polyline corner

fits Bezier segment into the polyline corners, using the binary search to find the larger fitting segments, see Fig 4. While analyzing performance of edge routing in MSAGLJS, we noticed that for a graph with more than 10000 edges these heuristics become the major bottleneck.

The reason for this was that we queried if a curve intersects any node of the whole graph. In spite of optimizing these operations with R-Trees [13], about %90 of the edge routing running time was spent on them. In addition, when the naive shortcutting of polyline corners fails the resulting path is not visually appealing, as shown in Fig. 3.

We replace global inefficient calculations with faster local procedures.

### Local path optimization

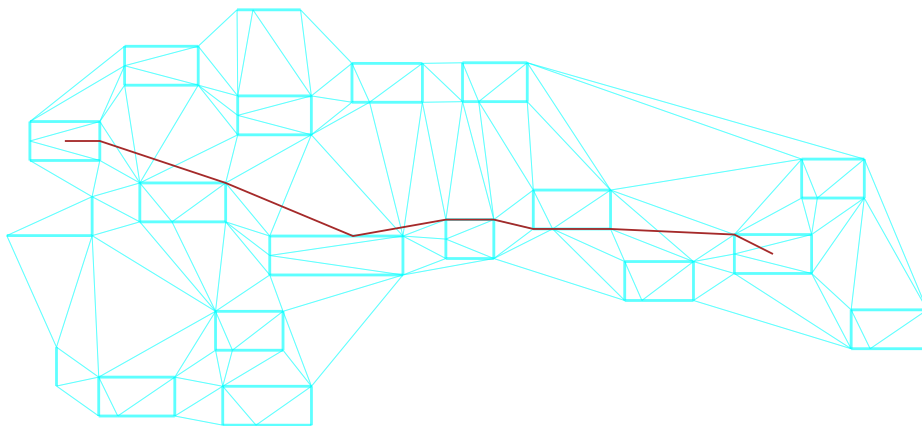
The idea of using a local optimization for paths is not a new one. The authors of [14] used it for hierarchical layouts, where the polygon containing the path is available. They write: "If  $\mathcal{P}$  does not contain holes ... we can apply a standard "funnel" algorithm [15, 16] for finding Euclidean shortest paths in a simple polygon". Let us describe our approach.

We call obstacles  $\mathcal{O}$  the set of polygons covering the original nodes, see Fig. 2. Before routing edges we calculate a Constrained Delaunay Triangulation [17] on  $\mathcal{O}$  and call it  $\mathcal{T}$ . Then for each edge of the graph we proceed with the following steps.

We route a path on the spanner, as illustrated by Fig. 5. Let us call this path  $\mathcal{L}$ . We would like to find a polygon,  $\mathcal{P}$ , containing  $\mathcal{L}$  and suitable to apply the funnel algorithm mentioned above. To obtain  $\mathcal{P}$ , let us consider  $\mathcal{U}$ , the set of all triangles from  $\mathcal{T}$  intersecting  $\mathcal{L}$ , and residing either outside of obstacles or inside on an obstacle containing one of the path ends, see Fig. 6. The union of  $\mathcal{U}$  gives us  $\mathcal{P}$ , and its boundary can be the set of all edges of  $\mathcal{U}$  that adjacent to only one of triangles from  $\mathcal{U}$ .

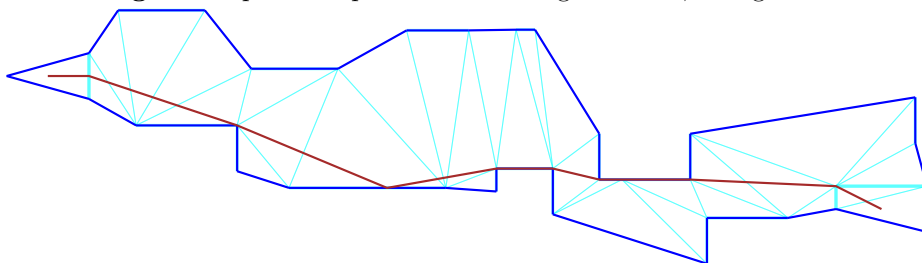
### References

1. "Graphexp." <https://github.com/bricaud/graphexp>.
2. "Graphviz." <http://www.graphviz.org/>.



54

**Fig. 5.** Unoptimized path  $\mathcal{L}$  with triangulation  $\mathcal{T}$ , a fragment.



55

**Fig. 6.** Polygon  $\mathcal{P}$  containing  $\mathcal{L}$ .

- 70 3. “Regraph.” <https://cambridge-intelligence.com/regraph/>.
- 71 4. “Skewed.” <https://graph-tool.skewed.de>.
- 72 5. “Circos.” <http://circos.ca/>.
- 73 6. H. Gibson, J. Faith, and P. Vickers, “A survey of two-dimensional graph layout  
74 techniques for information visualisation,” *Information visualization*, vol. 12, no. 3-  
75 4, pp. 324–357, 2013.
- 76 7. O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma, “What would a graph look like in this  
77 layout? a machine learning approach to large graph visualization,” *IEEE transac-  
78 tions on visualization and computer graphics*, vol. 24, no. 1, pp. 478–488, 2017.
- 79 8. Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. Chau, “Interactive multi-  
80 resolution exploration of million node graphs,” in *IEEE VIS*, 2013.
- 81 9. “Cosmograph.” <https://cosmograph.app>.
- 82 10. T. Dwyer and L. Nachmanson, “Fast edge-routing for large graphs,” in *Graph  
83 Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September  
84 22-25, 2009. Revised Papers 17*, pp. 147–158, Springer, 2010.
- 85 11. B. Flinchbaugh and L. Jones, “Strong connectivity in directional nearest-neighbor  
86 graphs,” *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 4, pp. 461–463,  
87 1981.
- 88 12. A. C.-C. Yao, “On constructing minimum spanning trees in k-dimensional spaces  
89 and related problems,” *SIAM Journal on Computing*, vol. 11, no. 4, pp. 721–736,  
90 1982.
- 91 13. A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Pro-  
92 ceedings of the 1984 ACM SIGMOD international conference on Management of  
93 data*, pp. 47–57, 1984.
- 94 14. D. P. Dobkin, E. R. Gansner, E. Koutsofios, and S. C. North, “Implementing a  
95 general-purpose edge router,” in *Graph Drawing: 5th International Symposium,  
96 GD’97 Rome, Italy, September 18–20, 1997 Proceedings 5*, pp. 262–271, Springer,  
97 1997.
- 98 15. B. Chazelle, “A theorem on polygon cutting with applications,” in *23rd Annual  
99 Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 339–349, IEEE,  
100 1982.
- 101 16. J. Hershberger and J. Snoeyink, “Computing minimum length paths of a given  
102 homotopy class,” *Computational geometry*, vol. 4, no. 2, pp. 63–97, 1994.
- 103 17. B. Delaunay, “Sur la sphere vide, bull. acad. science ussr vii: Class,” *Sci. Mat. Nat*,  
104 pp. 793–800, 1934.