

# 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>

1     **Abstract.** There has been progress in visualization of large graphs re-  
2     cently. Still, interacting with a large graph in the browser with the same  
3     ease as browsing an online map, inspecting the high level structure and  
4     zooming to the lower details, is still an unsolved problem, in our opinion.  
5     In this paper we describe MSAGLJS’s approach to two aspects of this  
6     problem. Firstly, we give a novel algorithm for edge routing, where the  
7     edges do not overlap the nodes. The algorithm does not necessarily cre-  
8     ates optimal paths but is efficient and creates visually appealing paths.  
   Secondly, to facilitate graph vizualization with DeckGL, we propose a  
   new simple and efficient approach to tiling. The aproch guarantees that  
   in every view the number of visible entities is not larger than a predefined  
   bound.

## 9    Introduction

## 10   Related work

11   Links to large graph visualization

12     [1]

13     [2]

14     [3]

15     [4]

16     [5]

17     [6]

18     machine learing approach [7]

19     [8]

20     [9]

## 21   Edge routing

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

34 Yao graph, which was introduced independently by Flinchbaugh and Jones [11]  
 35 and Yao [12]. This kind of graph is defined by the set of cones with the apices at  
 36 the vertices. The cones have the same angle, usually in the form of  $\frac{2\pi}{n}$ , where  $n$   
 37 is a natural number, and. The family of cones with the apex at a specific vertex  
 38 partition the plane as illustrated in Fig. 1. For each cone at most one edge is  
 39 created connecting the cone apex with a vertex inside of the cone, so the graph  
 40 has  $O(n)$  edges where  $n$  is the number of vertices.

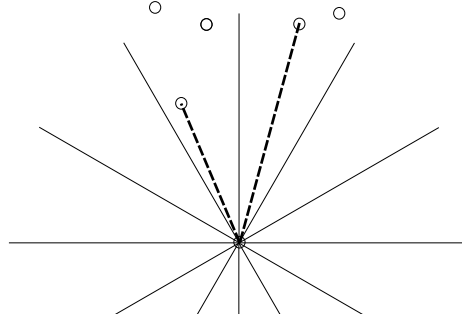
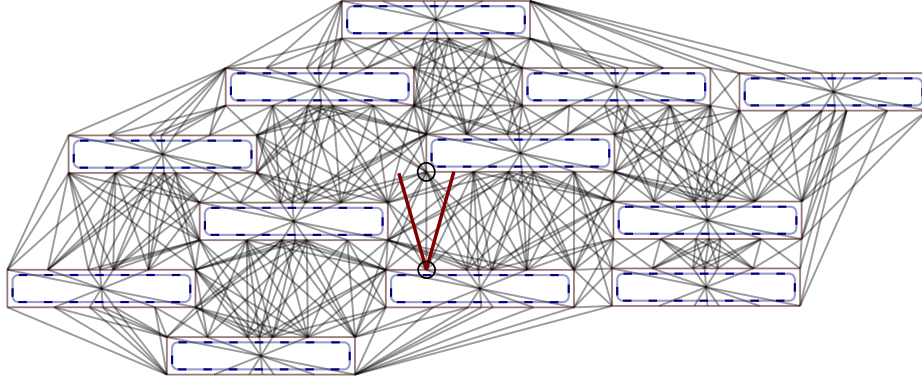
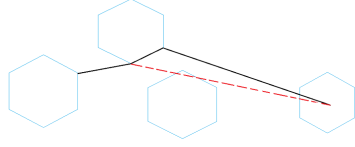


Fig. 1. Yao graph

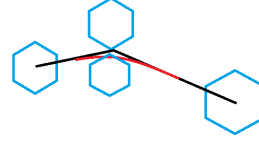


23 **Fig. 2.** Spanner graph is built using the idea of Yao graphs. The dashed curves are the  
 24 original node boundaries. Each original curve is surrounded by a polygon with some  
 25 offset to allow the polyline paths smoothing without intersecting the former.  
 26 The edge marked by the circles 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  $\frac{1}{\cos(\frac{\pi}{6})} \simeq 1.155$ .



**Fig. 3.** Unsuccessful shortcut



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

The approach of [10] first builds a polyline path through the spanner, then applies some local modifications to shorten and smoothen the path. It tries to shortcut a vertex iteratively, as illustrated in Fig 3. To smoothen it fits Bezier segments 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 1000 nodes these heuristics sometimes create a performance bottleneck in spite of using R-Trees[13]. In addition, when the naive shortcutting of polyline corners fails, the resulting path is not visually appealing, as shown in Fig. 3.

We replace these heuristics with a more precise optimization.

## Path optimization

Remember that a simple polygon is a polygon without holes.

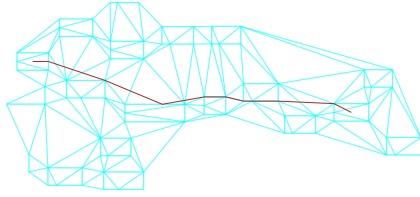
The application of the 'path in a simple polygon' optimization is not a new approach. The authors of [14] used it, but only for hierarchical layouts, where a simple polygon,  $\mathcal{P}$ , containing the path is available. They write: "If  $\mathcal{P}$  does not contain holes ... we can apply a standard "funnel" algorithm [15, 16] for finding Euclidean shortest paths in a simple polygon". In general case, for a non-layered layout, they build the visibility graph which is very expensive.

Here we show how to build polygon  $\mathcal{P}$ , and create a better path, for any layout. Let us describe our method.

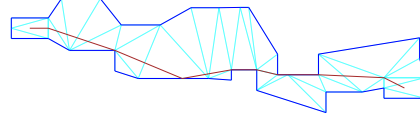
We call obstacles  $\mathcal{O}$  the set of polygons covering the original nodes, see Fig. 2. Before routing edges we calculate a Constrained Delaunay Triangulation [17] on  $\mathcal{O}$  and call it  $\mathcal{T}$ . Then for each edge of the graph we proceed with the following steps.

We route a path on the spanner, as illustrated by Fig. 5. Let us call this path  $\mathcal{L}$ . Let  $\mathcal{S}$  be the obstacle containing  $\mathcal{L}$ 's start point, and  $\mathcal{E}$  be the obstacle containing  $\mathcal{L}$ 's end point. To obtain  $\mathcal{P}$ , let us consider  $\mathcal{U}$ , the set of all triangles  $t \in \mathcal{T}$  such that either  $t \subset \mathcal{S} \cup \mathcal{E}$ , or  $t$  intersects  $\mathcal{L}$  and is not inside of any obstacle in  $\mathcal{O} \setminus \{\mathcal{S}, \mathcal{E}\}$ . The union of  $\mathcal{U}$  gives us  $\mathcal{P}$ . The boundary of  $\mathcal{P}$  comprises all edges  $e$  of the triangles from  $\mathcal{U}$  such that  $e$  is adjacent to exactly one triangle from  $\mathcal{U}$ , see Fig. 6.

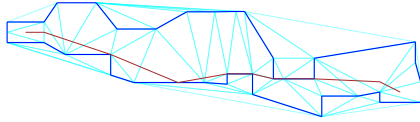
To apply the funnel algorithm [15, 16], we need to have a triangulation of  $\mathcal{P}$  such



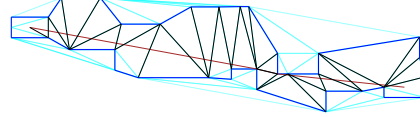
69 **Fig. 5.** Path  $\mathcal{L}$  with  $\mathcal{T}$ , a fragment.



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



71 **Fig. 7.** New triangulation of  $\mathcal{P}$ .



72 **Fig. 8.** The optimized path together  
73 with the sleeve diagonals.

84 that every edge of the triangulation is either a boundary edge of  $\mathcal{P}$ , or a diagonal  
85 of  $\mathcal{P}$ . In our situation  $\mathcal{U}$  usually does not have this property. We create a new  
86 Constrained Delaunay Triangulation of  $\mathcal{P}$ , where the set of constrained edges is  
87 the boundary of  $\mathcal{P}$ , see Fig. 7.

88 Finally, we apply the funnel algorithm with the new triangulation and obtain  
89 the path which is the shortest in the homotopy class of  $\mathcal{L}$ , as in Fig. 8.

90 The polygon  $\mathcal{P}$  is not necessarily simple, as shown in Fig. 9. In this example  
91 the path that we calculate with the funnel algorithm is not the shortest path  
92 inside of  $\mathcal{P}$ .

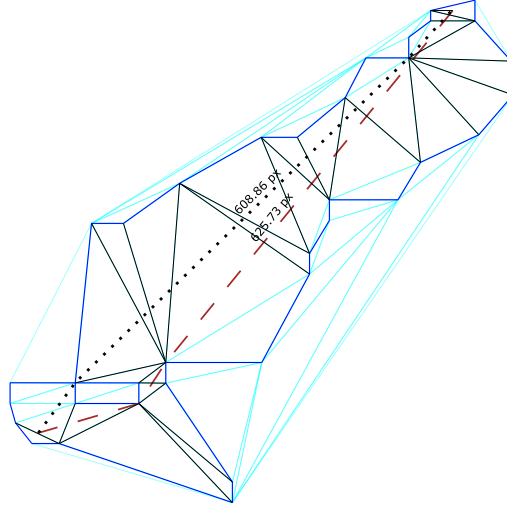
### 93 **Performance and quality comparison**

97 In Fig. 10 we compare the paths generated by the old and the new method. We  
98 can see that the paths produced by the new method have no kinks. We also  
99 know that these paths are the shortest in their 'channels'. Arguably, the new  
100 method produces better paths.

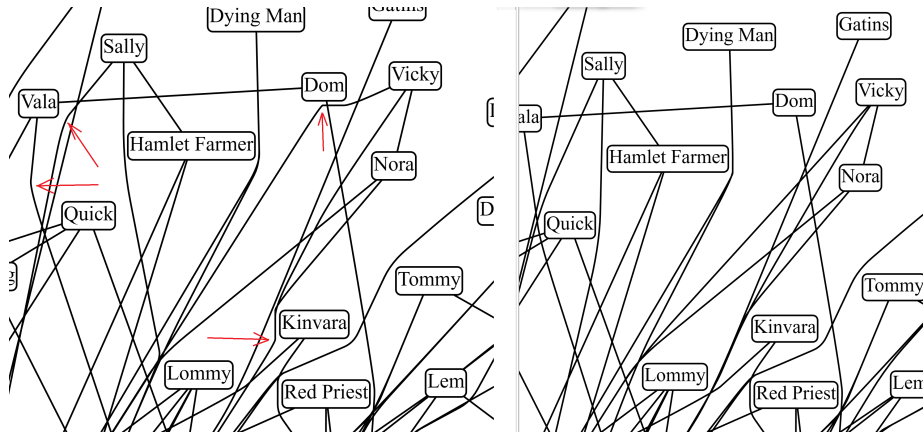
112 Our performance experiments are summarized in Table. 1. We see that the  
113 older approach outperforms the new one on the smaller graphs; those with the  
114 number of nodes under 2000. The new method is faster on the rest of the graphs.  
115 We still prefer to use the new method independently of the graph size since the  
116 total slowdown is insignificant, under a half second in our experiments, but the  
117 quality of the paths is better. On the larger graphs the new method runs faster  
118 and produces better paths, so it is an obvious choice.

### 119 **References**

- 120 1. "Graphexp." <https://github.com/bricaud/graphexp>.
- 121 2. "Graphviz." <http://www.graphviz.org/>.
- 122 3. "Regraph." <https://cambridge-intelligence.com/regraph/>.



74 **Fig. 9.**  $\mathcal{P}$  is not simple. The dotted path is shorter than the dashed one that  
 75 was found by the routing.



94 **Fig. 10.** The difference in the paths between the old, on the left, and the new,  
 95 on the right, paths. The arrows on the left fragment point to the kinks that were  
 96 removed by the new method.

| graph                    | nodes | edges  | old method's time | new time |
|--------------------------|-------|--------|-------------------|----------|
| social network [18]      | 407   | 2639   | 1.0               | 1.4      |
| b103 [19]                | 944   | 2438   | 1.6               | 2.0      |
| b100 [20]                | 1463  | 5806   | 5.6               | 5.785    |
| composers [21]           | 3405  | 13832  | 510.5             | 17.5     |
| p2p-Gnutella04 [22]      | 10876 | 39994  | 375.4             | 293.8    |
| facebook_combined [23]   | 4039  | 88234  | 132.2             | 119.1    |
| lastfm_asia_edges [24]   | 7626  | 27807  | 43.3              | 41.4     |
| deezer_europe_edges [24] | 28283 | 92753  | 1596.9            | 1209.3   |
| ca-HepPh [25]            | 12008 | 237010 | 521.2             | 495.0    |

**Table 1.** Performance comparison with time in seconds.

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

- 157 18. A. Beveridge and M. Chemers, “The game of game of thrones: Networked con-  
158 cordances and fractal dramaturgy,” in *Reading Contemporary Serial Television*  
159 *Universes*, pp. 201–225, Routledge, 2018.
- 160 19. “b103.” [https://github.com/microsoft/automatic-graph-](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b103.dot)  
161 [layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b103.dot](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b103.dot).
- 162 20. “b100.” [https://github.com/microsoft/automatic-graph-](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b100.dot)  
163 [layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b100.dot](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b100.dot).
- 164 21. “Skewed.” <http://mozart.diei.unipg.it/gdcontest/contest2011/composers.xml>.
- 165 22. “p2p-gnutella04.” <https://snap.stanford.edu/data/p2p-Gnutella04.html>.
- 166 23. “facebookcombined.” [https://snap.stanford.edu/data/facebook\\_combined.txt.gz](https://snap.stanford.edu/data/facebook_combined.txt.gz).
- 167 24. B. Rozemberczki and R. Sarkar, “Characteristic Functions on Graphs: Birds of a  
168 Feather, from Statistical Descriptors to Parametric Models,” in *Proceedings of the*  
169 *29th ACM International Conference on Information and Knowledge Management*  
170 *(CIKM ’20)*, p. 1325–1334, ACM, 2020.
- 171 25. J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graph evolution: Densification  
172 and shrinking diameters,” *ACM transactions on Knowledge Discovery from Data*  
173 *(TKDD)*, vol. 1, no. 1, pp. 2–es, 2007.