

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. However, interacting with a large graph in an Internet browser
3 with the same ease as browsing an online map, inspecting the high level
4 structure and zooming in to the high level detail, is still an unsolved
5 problem. In this paper we describe novel approaches to several aspects
6 of this problem.
7 We give a new algorithm for edge routing, where the edges do not overlap
8 the nodes. The algorithm does not necessarily creates the optimal paths,
9 but is efficient and creates visually appealing routes.
To facilitate graph vizualization with DeckGL, we propose a new simple
and fast tiling method. The method guarantees that in every view, except
of the highest layer, the number of visible entities is not larger than a
predefined bound. Our method provides a high level overview of the
graph. The edge routing algorithm mentioned above is reused at the
tiling stage to simplify the paths on the lower levels. In addition, we
bundle edges per-tile as an optimization heuristic.

10 Introduction

11 We discuss large but not huge graphs. The maximum number of vertices of
12 graphs we looked at was 28283, and the maximum number of edges was 237010.
13 There are many algorithms that calculate a node layout for such graphs in a few
14 seconds [1, 2], and we do not discuss them.

15 In the first part of the paper we address edge routing where an edge only
16 intersects the nodes it is adjacent to. Our approach works for any node layout,
17 as long as the nodes do not overlap each other. The approach builds on [3] and
18 improves it.

19 Related work

20 [4]
21 [5]
22 [6]

23 [7]
 24 [8]
 25 machine learning approach [9]
 26 [10]
 27 [11]

28 Edge routing

39 The edge routing starts, as in [3], by building a spanner graph, an approximation
 40 of the full visibility graph. The spanner, see Fig. 2, is built on a variation of a
 41 Yao graph, which was introduced independently by Flinchbaugh and Jones [12]
 42 and Yao [13]. This kind of graph is defined by the set of cones with the apices at
 43 the vertices. The cones have the same angle, usually in the form of $\frac{2\pi}{n}$, where n
 44 is a natural number, and. The family of cones with the apex at a specific vertex
 45 partition the plane as illustrated in Fig. 1. For each cone at most one edge is
 46 created connecting the cone apex with a vertex inside of the cone, so the graph
 47 has $O(n)$ edges where n is the number of vertices.

52 The approach of [3] first builds a polyline path through the spanner, then
 53 applies some local modifications to shorten and smoothen the path. It tries to
 54 shortcut a vertex iteratively, as illustrated in Fig 3. To smoothen it fits Bezier seg-
 55 ments into the polyline corners, using the binary search to find the larger fitting
 56 segments, see Fig 4. While analyzing performance of edge routing in MSAGLJS,
 57 we noticed that for a graph with more than 1000 nodes these heuristics some-
 58 times create a performance bottleneck in spite of using R-Trees[14].
 59 In addition, when the naive shortcutting of polyline corners fails, the resulting
 60 path is not visually appealing, as shown in Fig. 3.

61 We replace these heuristics with a more precise optimization.

62 Path optimization

63 Remember that a simple polygon is a polygon without holes.

64 An application of the 'path in a simple polygon' optimization is not a new
 65 approach. The authors of [15] used it, but only for hierarchical layouts, where a
 66 simple polygon, \mathcal{P} , containing the path is available. They write: "If \mathcal{P} does not
 67 contain holes ... we can apply a standard "funnel" algorithm [16, 17] for finding
 68 Euclidean shortest paths in a simple polygon". In general case, for a non-layered
 69 layout, they build the visibility graph which is very expensive.

70 Here we drop the requirement that \mathcal{P} is simple. Indeed, to run the "funnel"
 71 algorithm one only needs a sleeve: a sequence of triangles adjoined on an edge
 72 and leading from the start to the end of the path. We show how to build polygon
 73 \mathcal{P} , create a sleeve, and produce an optimized path, for any layout. Let us describe
 74 our method.

75 We call obstacles \mathcal{O} the set of polygons covering the original nodes, see Fig. 2.
 76 Before routing edges we calculate a Constrained Delaunay Triangulation [18] on

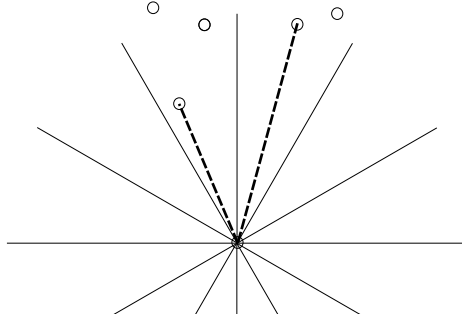


Fig. 1. Yao graph

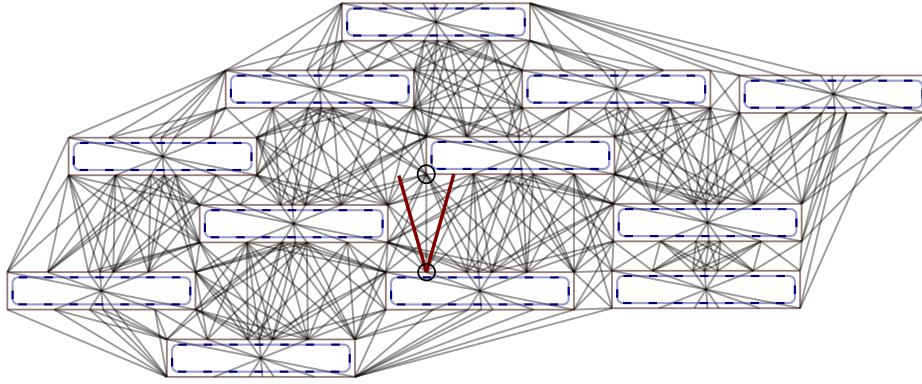


Fig. 2. Spanner graph is built using the idea of Yao graphs. The dashed curves are the original node boundaries. Each original curve is surrounded by a polygon with some offset to allow the polyline paths smoothing without intersecting the former. The edge marked by the circles is created because the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

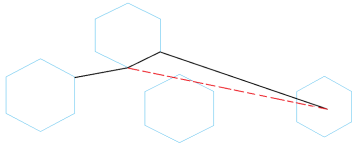


Fig. 3. Unsuccessful shortcut

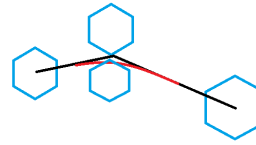
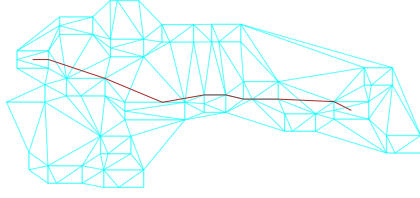
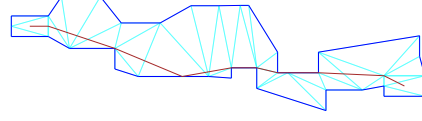


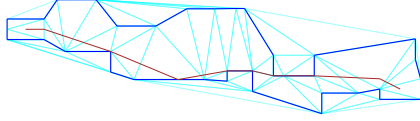
Fig. 4. Fitting a Bezier segment into a polyline corner



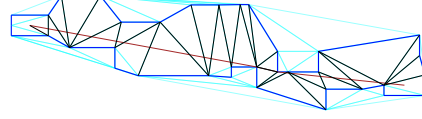
79 **Fig. 5.** Path \mathcal{L} with \mathcal{T} , a fragment.



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



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



82 **Fig. 8.** The optimized path together
83 with the sleeve diagonals.

77 \mathcal{O} and call it \mathcal{T} . Then for each edge of the graph we proceed with the following
78 steps.

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

92 To create the sleeve [16, 17], we need to have a triangulation of \mathcal{P} such that
93 every edge of the triangulation is either a boundary edge of \mathcal{P} , or a diagonal of
94 \mathcal{P} . In our setup \mathcal{U} might not have this property, as in Fig. 6. We create a new
95 Constrained Delaunay Triangulation of \mathcal{P} , where the set of constrained edges is
96 the boundary of \mathcal{P} , see Fig. 7.

97 We trace \mathcal{L} through the new triangulation and obtain the sleeve. Finally, we
98 apply the funnel algorithm on the sleeve and obtain the path which is the shortest
99 in the homotopy class of \mathcal{L} , as illustrated in Fig. 8.

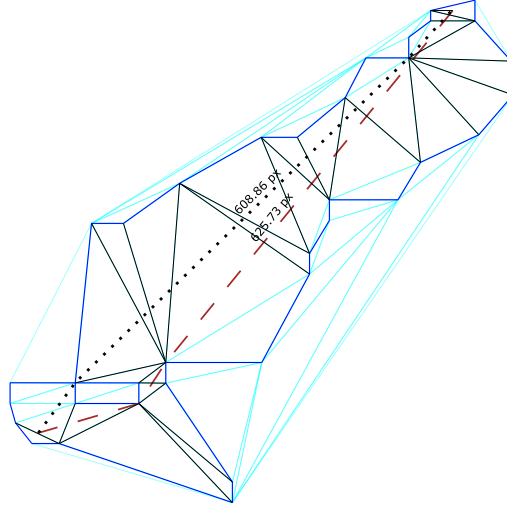
100 The discussion [19] of the algorithm helped us in the implementation.

101 Polygon \mathcal{P} is not necessarily simple, as shown in Fig. 9. In this example the
102 path that we calculate with the funnel algorithm is not the shortest path inside
103 of \mathcal{P} .

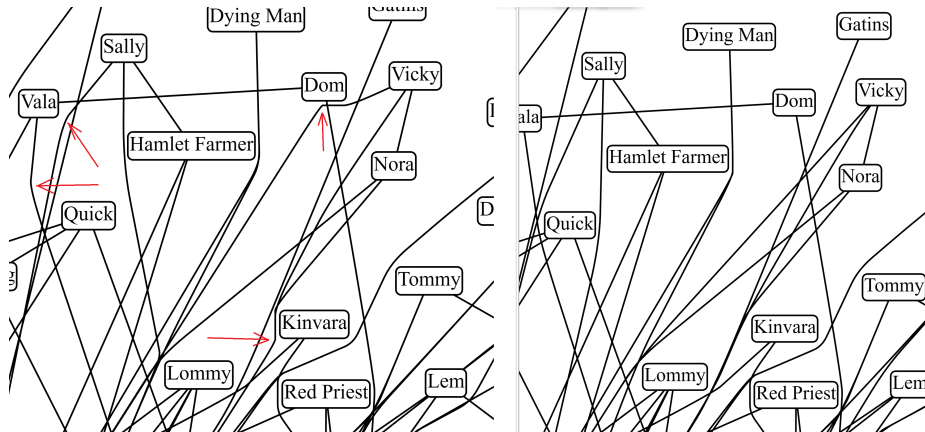
104 Performance and quality comparison

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

123 Our performance experiments are summarized in Table. 1. We see that the
124 older approach outperforms the new one on the smaller graphs; those with the



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



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

number of nodes under 2000. The new method is faster on the rest of the graphs. We still prefer to use the new method independently of the graph size since the total slowdown is insignificant, under a half second in our experiments, but the quality of the paths is better. On the larger graphs the new method runs faster and produces better paths, so it is an obvious choice.

graph	nodes	edges	old method's time	new time
social network [20]	407	2639	1.0	1.4
b103 [21]	944	2438	1.6	2.0
b100 [22]	1463	5806	5.6	5.785
composers [23]	3405	13832	510.5	17.5
p2p-Gnutella04 [24]	10876	39994	375.4	293.8
facebook_combined [25]	4039	88234	132.2	119.1
lastfm_asia_edges [26]	7626	27807	43.3	41.4
deezer_europe_edges [26]	28283	92753	1596.9	1209.3
ca-HepPh [27]	12008	237010	521.2	495.0

Table 1. Performance comparison with time in seconds.

129

1 Tiling

The algorithm works in two phases. The first phase builds more and more detailed levels with smaller tiles until no more tile subdivision is required. Then second phase goes from the higher to lower levels and finalizes the levels.

A tile is a pair of a rectangle and data (*rect*, *tile_data*). Keys to the tile hierarchy are in the form (i, j, z) , where z is the level index and pair (i, j) indicates the rectangle inside of the level. The initial, the tile with the largest rectangle on level 0 is represented by the triplet $(0, 0, 0)$. For $z = 1$ there are four tiles $(0, 0, 1)$, $(0, 1, 1)$, $(1, 0, 1)$ and $(1, 1, 1)$. Each tile (i, j, z) can be subdivided into four tiles of the same size one level higher: $(2i, 2j, z + 1)$, $(2i, 2j + 1, z + 1)$, $(2i + 1, 2j, z + 1)$, and $(2i + 1, 2j + 1, z + 1)$.

Each z -level is represented by a map $L(z)$, so $L(z)(i, j)$ gives us a specific tile. During the first phase we can discover some empty tiles which correspond to $L(z)(i, j)$ being not defined.

The tiling works when the edge routing is done, so each edge e has an associated curve $c(e)$. During the subdivision process we create pairs *curve clips*, (e, p) , where p is $c(e)$ or a continuous trimmed piece of $c(e)$. By construction we will have the property that for each curve clip (e, p) the curve p belong to the corresponding tile rectangle and it might touch the boundary of rectangle only at the endpoints of p .

One of the parameters controlling the algorithm is the number for tile capacity, \mathcal{C} , setting the upper limit on how many elements can be visible in one

152 tile. The elements could be a curve clip, an arrowhead, a node, or a label. In our
 153 setting \mathcal{C} is set by default to 10000.

154 The first phase starts with $L(0) = \{(0,0) \rightarrow \text{tile data}\}$: the map consisting
 155 of only one lowest tile, and the elements of *tile data* are all curve clips $e, c(e)$,
 156 all graph nodes, all edge labels, and all edge arrowheads. If the total number of
 157 these elements is less than \mathcal{C} then the first phase stops; this is the usual case for
 158 a small graph.

159 If it is not the case then the first phase continues working. Let us suppose
 160 that the current level is z . We denote by $C(i, j)$ the number of elements in
 161 $L(z)(i, j)$, in other words, the number elements crossing tile (i, j, z) .

162 For the minimal size of the tile we take $(8 \times w, 8 \times h)$, where w is the average
 163 width and h is the average height of the nodes of the graph. The algorithm starts
 164 after the edge routing is done, so each edge has a curve, an optional label, and
 165 arrowheads associated with it. The algorithm keeps a map from tilesInitially, we
 166 create one top level tile and

167 References

- 168 1. Y. Hu and L. Shi, “Visualizing large graphs,” *Wiley Interdisciplinary Reviews:*
 169 *Computational Statistics*, vol. 7, no. 2, pp. 115–136, 2015.
- 170 2. U. Brandes and C. Pich, “Eigensolver methods for progressive multidimensional
 171 scaling of large data,” in *Graph Drawing: 14th International Symposium, GD*
 172 *2006, Karlsruhe, Germany, September 18-20, 2006. Revised Papers 14*, pp. 42–
 173 53, Springer, 2007.
- 174 3. T. Dwyer and L. Nachmanson, “Fast edge-routing for large graphs,” in *Graph*
 175 *Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September*
 176 *22-25, 2009. Revised Papers 17*, pp. 147–158, Springer, 2010.
- 177 4. “Graphviz.” <http://www.graphviz.org/>.
- 178 5. “Regraph.” <https://cambridge-intelligence.com/regraph/>.
- 179 6. “Skewed.” <https://graph-tool.skewed.de>.
- 180 7. “Circos.” <http://circos.ca/>.
- 181 8. H. Gibson, J. Faith, and P. Vickers, “A survey of two-dimensional graph layout
 182 techniques for information visualisation,” *Information visualization*, vol. 12, no. 3-
 183 4, pp. 324–357, 2013.
- 184 9. O.-H. Kwon, T. Crnovrsanin, and K.-L. Ma, “What would a graph look like in this
 185 layout? a machine learning approach to large graph visualization,” *IEEE transac-*
 186 *tions on visualization and computer graphics*, vol. 24, no. 1, pp. 478–488, 2017.
- 187 10. Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. Chau, “Interactive multi-
 188 resolution exploration of million node graphs,” in *IEEE VIS*, 2013.
- 189 11. “Cosmograph.” <https://cosmograph.app>.
- 190 12. B. Flinchbaugh and L. Jones, “Strong connectivity in directional nearest-neighbor
 191 graphs,” *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 4, pp. 461–463,
 192 1981.
- 193 13. A. C.-C. Yao, “On constructing minimum spanning trees in k-dimensional spaces
 194 and related problems,” *SIAM Journal on Computing*, vol. 11, no. 4, pp. 721–736,
 195 1982.
- 196 14. A. Guttman, “R-trees: A dynamic index structure for spatial searching,” in *Pro-*
 197 *ceedings of the 1984 ACM SIGMOD international conference on Management of*
 198 *data*, pp. 47–57, 1984.

- 199 15. D. P. Dobkin, E. R. Gansner, E. Koutsofios, and S. C. North, "Implementing a
200 general-purpose edge router," in *Graph Drawing: 5th International Symposium,*
201 *GD'97 Rome, Italy, September 18–20, 1997 Proceedings* 5, pp. 262–271, Springer,
202 1997.
- 203 16. B. Chazelle, "A theorem on polygon cutting with applications," in *23rd Annual*
204 *Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 339–349, IEEE,
205 1982.
- 206 17. J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given
207 homotopy class," *Computational geometry*, vol. 4, no. 2, pp. 63–97, 1994.
- 208 18. B. Delaunay, "Sur la sphere vide, bull. acad. science ussr vii: Class," *Sci. Mat. Nat*,
209 pp. 793–800, 1934.
- 210 19. "Funnel algorithm." <https://page.mi.fu-berlin.de/mulzer/notes/alggeo/polySP.pdf>.
- 211 20. A. Beveridge and M. Chemers, "The game of game of thrones: Networked con-
212 cordances and fractal dramaturgy," in *Reading Contemporary Serial Television*
213 *Universes*, pp. 201–225, Routledge, 2018.
- 214 21. "b103." [https://github.com/microsoft/automatic-graph-](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b103.dot)
215 [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).
- 216 22. "b100." [https://github.com/microsoft/automatic-graph-](https://github.com/microsoft/automatic-graph-layout/blob/master/GraphLayout/Test/MSAGLTests/Resources/DotFiles/LevFiles/b100.dot)
217 [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).
- 218 23. "Skewed." <http://mozart.diei.unipg.it/gdcontest/contest2011/composers.xml>.
- 219 24. "p2p-gnutella04." <https://snap.stanford.edu/data/p2p-Gnutella04.html>.
- 220 25. "facebookcombined." https://snap.stanford.edu/data/facebook_combined.txt.gz.
- 221 26. B. Rozemberczki and R. Sarkar, "Characteristic Functions on Graphs: Birds of a
222 Feather, from Statistical Descriptors to Parametric Models," in *Proceedings of the*
223 *29th ACM International Conference on Information and Knowledge Management*
224 *(CIKM '20)*, p. 1325–1334, ACM, 2020.
- 225 27. J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification
226 and shrinking diameters," *ACM transactions on Knowledge Discovery from Data*
227 *(TKDD)*, vol. 1, no. 1, pp. 2–es, 2007.