

Curvilinear Graph Drawing Using the Force-Directed Method

Benjamin Finkel¹ and Roberto Tamassia²

¹ MIT Lincoln Laboratory

`finkel@ll.mit.edu`

² Brown University

`rt@cs.brown.edu`

Abstract. We present a method for modifying a force-directed graph drawing algorithm into an algorithm for drawing graphs with curved lines. Our method is based on embedding control points as dummy vertices so that edges can be drawn as splines. Our experiments show that our method yields aesthetically pleasing curvilinear drawing with improved angular resolution. Applying our method to the GEM algorithm on the test suite of the “Rome Graphs” resulted in an average improvement of 46% in angular resolution and of almost 6% in edge crossings.

1 Introduction

Curvilinear drawings of graphs give a significant amount of flexibility to a layout algorithm, creating the potential for improved aesthetics. Such drawings are ideally suited for several applications (e.g., flight maps). However, the literature on curvilinear drawing algorithms is not as extensive as that on straight-line and orthogonal drawings. In this paper, we present a methodology for computing curvilinear drawings using force-directed methods and we report on the result of experiments showing that our technique yields aesthetically pleasing curvilinear drawings with improved angular resolution and number of crossings.

The idea of the force-directed method is to use physical simulations to lay out a graph. Forces are calculated, applied to vertices, and recalculated over many iterations. In the pioneering “spring-embedder” algorithm [10], the edges are modeled as stretchable springs of different length, which oscillate until the system reaches equilibrium. Effective force-directed techniques include subatomic forces [12] and simulated annealing [8]. An experimental comparison of force-directed methods is presented in [3]. Recent work includes [13, 19].

Previous work on curvilinear drawings of graphs has focused on planar drawings and on edge-routing methods. Early systems for layered drawings of digraphs use heuristics that transform a drawing with polygonal chains into a curvilinear drawing by replacing polygonal chains with splines [14–16]. The routing of a curvilinear edge that is being added to an existing drawing is investigated in [9]. In [17, 18], algorithms for drawing planar graphs using cubic Bézier curves for the edges are given. In [7], an algorithm is presented for drawing planar graphs

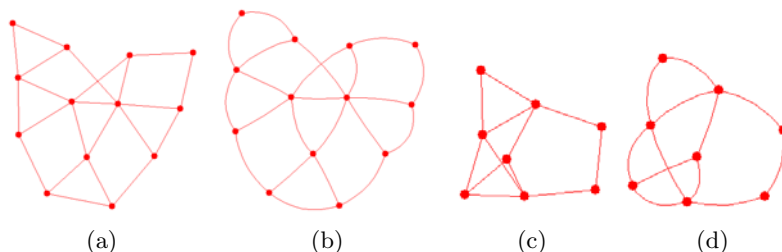


Fig. 1. Layout improvements obtained by our method: (a) straight-line drawing produced by the KK algorithm; (b) associated curvilinear drawing; (c) straight-line drawing produced by the GEM algorithm; (d) associated curvilinear drawing.

with asymptotically optimal area and angular resolution such that the edges are sequences of two circular arcs. Brandes and Wagner show how to use Bézier curves to draw graphs with vertices at fixed locations [6], with applications to the display of geographic networks. Related work on 3D curvilinear drawings of geographic networks is presented in [4, 5, 21].

Imagine an architect who uses graph drawing software to design a sculpture garden. In the graph, vertices represent sculptures and edges represent paths. Using a standard force-directed algorithm, the architect would probably get a drawing with a good spread of sculptures, but a mediocre layout of paths. The architect might want the paths themselves to be more separated, and to arrive at the sculptures at more distinct angles. These properties correspond to the aesthetic criteria of edge separation and angular resolution. It seems plausible that by introducing curved paths the architect could improve the aesthetic layout with respect to these two qualities. The architect might also prefer smooth, curved lines to the rigid lines connecting the sculptures in the layout generated by the standard force-directed algorithm. We show in Fig. 1 how drawings produced by the KK [20] and GEM [11] algorithms can be smoothed and aesthetically improved using the method set forth in this paper.

The fundamental function of force-directed methods is to find a layout for the vertices, which are the only objects subject to forces. Edges merely influence these forces. In many circumstances, this is acceptable. However, one could imagine a situation in which the layout of the edges is also important and should be considered by the algorithm. Thus, we would like to give edges a “mass-like” quality so they can also be pushed and pulled and acted upon by forces.

2 Force-Directed Curvilinear Drawings

For the curvilinear drawings studied in this paper, we consider three relevant aesthetic criteria: angular resolution, edge separation, and number of crossings. The *angular resolution* refers to the angles formed by pairs of edges incident on a vertex. In general, small angles are not desirable. We measure the angular resolution as the difference between the smallest angle and the optimal angle at a vertex v ($360/\deg(v)$), averaged over all vertices. Thus, lower values in-

dicating better layouts. The *edge separation* refers to the distance between an edge and another non-incident, non-intersecting edge. In this paper, we measure the edge separation as the average distance between pairs of non-incident, non-intersecting edges. Hence, larger values are desirable. Finally, edge crossings are undesirable and should be minimized.

Our algorithm for curvilinear graph drawing using the force-directed method is based on the following simple idea (see Fig. 2a–b). Given a graph G , we insert a new fictitious vertex C along each edge (A, B) , thus replacing (A, B) with a path of two edges (A, C) and (C, B) . Let G' be the resulting graph. We compute a straight-line drawing of G' using a force-directed algorithm. Finally, we transform the drawing of G' into a drawing of G by replacing each polygonal chain (A, C, B) , where C is a fictitious vertex, with a curve that joins A and B and uses C as a control point. More generally, we can embed several fictitious vertices (control points) on each edge to magnify the curving effect. In practice, one or two vertices will produce good results. Our approach generalizes the one used by Brandes and Wagner [6] for the layout of geographic networks, where the vertex locations are fixed and forces operate only in local neighborhoods.

Adding vertices and edges to a graph may increase the number of crossings. In theory, one could prevent the addition of new crossings by imposing boundaries on the control vertices. However, most force-directed algorithms, while evaluating a modified graph, will sometimes generate new crossings (see Fig. 2c). In practice, many of these crossings occur among edges incident on the same vertex and can be easily detected.

Any number of algorithmic heuristics could be used to unwind these crossings. We use a binding technique, inspired from [6], which was designed to be simple and algorithm independent: its only mechanism is adding hidden edges. To bind a vertex, connect the closest control vertices around that vertex in the order of the “real” (non-control) endpoints. The points are sorted based on polar coordinates, similar to a Graham Scan. The vertices are connected with edges that will not be drawn, but will pull the control points into place. Convex angles are also excluded, so as to preserve good angular resolution.

Binding a vertex will generally fix these added local intersections. It should be noted, however, that binding is specific to each embedding of a graph. Thus, it can be done only after the algorithm has run and then it requires the algorithm to be repeated from the current position with the new hidden edges. In this way, the bindings can be done in several passes, or all at once, yielding different results.

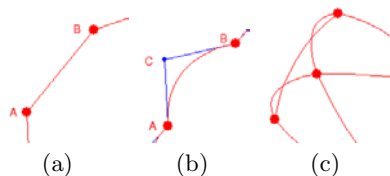


Fig. 2. (a–b) Inserting a fictitious vertex C , serving as a control point, on edge (A, B) . (c) Crossing between edges incident on the same vertex.

3 Implementation and Experimental Results

We have implemented our force-directed curvilinear graph drawing method in Java using the JDSL library [1] and the VGJ tool [2]. Experiments were performed on 1.5 GHz AMD Athlon workstations with 512MB RAM running Linux. The test suite was made from the “Rome Graphs” obtained from graphdrawing.org. The force-directed algorithms chosen were GEM [11] and KK [20] because of their effectiveness, speed and simplicity. Indeed, GEM and KK were shown to be all-around performers in [3]. The implementations of GEM and KK were adapted from the JDSL library and VGJ tool, respectively.

We use a Bézier curve because of its attractive shape and geometric properties: it is contained inside the convex hull of its endpoints and control points, and its slope starts directly towards a control point. Edges with single and double control points use quadratic and cubic Bézier curves, respectively. For binding, there are two successive passes. We also test the effect of binding all vertices.

A step-by-step example of the execution of the modified GEM method using quadratic Bézier curves and the “bind all” heuristic is shown in Fig. 3.

Each one of the Rome Graphs was drawn twice, with and without the curvilinear method, starting from the same initial random vertex placement. Fictitious control vertices were evenly spaced on the edges. The above process was repeated with three different binding heuristic options. First with no binding, simply executing the algorithm on the modified graph and measuring the aesthetics. Second, with the binding heuristic on only vertices with local intersections. In this case we ran the algorithm on the graph as before, then bound vertices

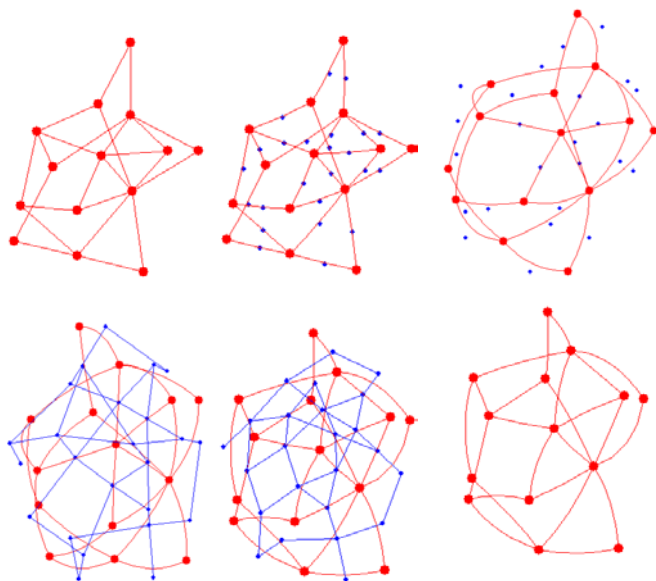


Fig. 3. Step-by-step example of execution of the modified GEM method using quadratic Bézier curves and the “bind all” heuristic.

Table 1. Averages of the aesthetic properties using our curvilinear method with the GEM force-directed layout algorithm on the Rome Graphs.

edge type, type of binding	angular resolution	edge separation	# crossings
straight edges (without method)	46.175257	0.043419	30.718649
quadratic Bézier, none	34.447798	0.044492	29.652424
quadratic Bézier, crossed	32.612817	0.044642	30.316422
quadratic Bézier, all	31.748013	0.045090	30.393090
cubic Bézier, none	31.233901	0.044849	29.209033
cubic Bézier, crossed	28.199770	0.045294	29.216517
cubic Bézier, all	24.814479	0.045931	28.986946

with local intersections, executed, bound again, and executed a final time before making the measurements. Finally, a test was done where the algorithm was executed on the graph, then all vertices were bound, and then the algorithm was run again.

Our experimental results, summarized in Table 1, show that the curvilinear drawing method significantly improves the angular resolution. Binding all vertices, using cubic Bézier curves led to an overall average angular resolution 46% closer to optimal and average decrease of more than one crossing per graph.

It should be noted that binding in general was not very effective when using quadratic Bézier edges. This is because two vertices share each control point; binding from one vertex can adversely affect the other. For cubic Bézier curves, the results show that binding is an extremely effective heuristic; it is best when used on all vertices, not just the ones with local intersections.

Unlike the GEM algorithm, the KK algorithm did not perform well with our curvilinear method. Since the method increases the number of vertices and edges, it affects the running time of the drawing algorithm. Because of its $O(n^3)$ running time per iteration, compared to GEM's $O(n^2)$, KK did not scale well and was prohibitively slow on large graphs. In addition, the KK algorithm uses graph-theoretic distances, and thus the presence of the fictitious vertices hampers the effectiveness of the algorithm itself, creating far too many new crossings to justify any improvement in angular resolution or edge separation.

Future research should investigate the effect of this process on other force-directed methods, such as [8, 13, 19]. It would also be interesting to investigate the use of interpolating splines (e.g., Catmull-Rom), which go through their control points, instead of Bézier curves.

Acknowledgements

We would like to thank Mike Goodrich and Franco Preparata for useful discussions. Our research was supported in part by National Science Foundation grants CCR-0098068 and DUE-0231202. This work originates from Benjamin Finkel's Undergraduate Honors Thesis and was performed while he was at Brown University.

References

1. JDSL: the data structures library in Java. <http://jdsl.org>.
2. VGJ: Visualizing graphs with Java. http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html.
3. F. J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In *Graph Drawing (Proc. GD 1995)*, LNCS 1027, pp. 76–87, 1996.
4. U. Brandes, G. Shubina, and R. Tamassia. Improving angular resolution in visualizations of geographic networks. In *Proc. Joint Eurographics — IEEE TCVG Symposium on Visualization (VisSym '00)*, pp. 23–32, 2000.
5. U. Brandes, G. Shubina, R. Tamassia, and D. Wagner. Fast layout methods for timetable graphs. In *Graph Drawing (Proc. GD 2000)*, LNCS 1984, pp. 127–138, 2001.
6. U. Brandes and D. Wagner. Using graph layout to visualize train interconnection data. In *Graph Drawing (Proc. GD 1998)*, LNCS 1547, pp. 44–56, 1998.
7. C. C. Cheng, C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Drawing planar graphs with circular arcs. In *Graph Drawing (Proc. GD 1999)*, LNCS 1731, pp. 117–126, 1999.
8. R. Davidson and D. Harel. Drawing graphics nicely using simulated annealing. *ACM Trans. Graph.*, 15(4):301–331, 1996.
9. D. P. Dobkin, E. R. Gansner, E. Koutsofios, and S. C. North. Implementing a general-purpose edge router. In *Graph Drawing (Proc. GD 1997)*, LNCS 1353, pp. 262–271, 1997.
10. P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.
11. A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Graph Drawing (Proc. GD 1994)*, LNCS 894, pp. 388–403, 1995.
12. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.
13. P. Gajer, M. T. Goodrich, and S. G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. *Graph Drawing (Proc. GD 2000)*, LNCS 1984, pp. 211–221, 2000.
14. E. R. Gansner, E. Koutsofios, S. C. North, and K. P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19:214–230, 1993.
15. E. R. Gansner, S. C. North, and K. P. Vo. DAG – A program that draws directed graphs. *Softw. – Pract. Exp.*, 18(11):1047–1062, 1988.
16. A. Garg and R. Tamassia. GIOTTO3D: A system for visualizing hierarchical structures in 3D. In *Graph Drawing (Proc. GD 1996)*, LNCS 1190, pp. 193–200, 1997.
17. M. T. Goodrich and C. G. Wagner. A framework for drawing planar graphs with curves and polylines. In *Graph Drawing (Proc. GD 1998)*, LNCS 1547, pp. 153–166, 1998.
18. C. Gutwenger and P. Mutzel. Planar polyline drawings with good angular resolution. In *Graph Drawing (Proc. GD 1998)*, LNCS 1547, pp. 167–182, 1998.
19. D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Graph Drawing (Proc. GD 2002)*, LNCS 2528, pp. 207–219, 2002.
20. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inform. Process. Lett.*, 31:7–15, 1989.
21. T. Munzner, E. Hoffman, K. Claffy, and B. Fenner. Visualizing the global topology of the MBone. In *Proc. IEEE Symp. on Information Visualization*, pp. 85–92, 1996.