Curvilinear Graph Drawing Using The Force-Directed Method

by
Benjamin Finkel
Sc. B., Brown University, 2003

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2003

This thesis by Benjamin Finkel is accepted in its present form by
the Department of Computer Science as satisfying the research requirement
for the awardment of Honors.

Date _____ _____

Roberto Tamassia, Reader

Date _____ _____

Franco Preparata, Reader

Acknowledgements

I would like to thank Roberto Tamassia for all his help and guidance over the past year.

Contents

Chapter 1

Introduction

In the field of graph drawing [1], one of the paramount problems is to draw graphs that satisfy a number of aesthetic criteria including graph area, number of crossings, number of bends, vertex resolution, angular resolution, and edge separation. Drawings which satisfy these criteria are often not only the most attractive, but also the most practical. Orthogonal graphs with few bends, few crossings and small area often represent an efficient design for a computer chip. And graphs with good vertex resolution, angular resolution, and edge separation might represent an effective design for a map.

A particularly unique and successful strategy for drawing graphs which meet many of these aesthetic criteria is the force directed method [8]. Unlike most algorithms rooted in theory, the force directed simulates a system of natural forces to find a graph layout.

Curvilinear drawings adds a significant amount of flexibility to a graph drawing, creating the potential for improved aesthetics. Many applications of graph drawing–paths, maps, charts–are designed for curved-line drawings. However, there are few curved-line drawing algorithms that benefit from the success of force directed methods.

The goal of this paper is to present a methodology for incorporating curved-line drawing into the force directed method, and to prove that, in general, it improves the overall aesthetic appeal of a graph.

Chapter 2

Previous Work

The principle 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. The "spring-embedder" is the original force directed method, where all the edges are modeled as stretchable springs of different length which oscillate until the system reaches equilibrium [6]. Other force-directed methods make physical models from subatomic forces of attraction and repulsion [8] and energy minimization using simulated annealing [5]. A survey of force directed methods can be found in [3].

Drawing graphs with curved edges has been previously researched. Brandes and Wagner explore the idea of using Bezier curves in graph drawing to display train interconnection data [4]. They use the Bezier control vertices as dummy vertices which are moved by repulsive and attractive forces, as done in this paper. However, they develop a specific algorithm where the vertex locations are fixed and forces operate only in local neighborhoods. They also provide no analytical data which can measure the improvements of the curved lines. This paper will generalize curvilinear graph drawing to work with any force directed layout algorithm, develop new heuristic improvements, and give experimental data to quantify its effect.

Chapter 3

Motivating Example

3.1 Force Directed Method's Limitations

The fundamental function of force directed methods is to find a layout for the vertices. They are the only objects with forces, directions, or ultimately something equivalent to mass. Edges merely act as forces between vertices which will affect their final position. For 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.

3.2 The Sculpture Garden

Imagine an architect who is using graph drawing software to design a new outdoor sculpture garden. In the graph, vertices are equated with sculptures and edges are equated with 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 are equated with edge separation and angular resolution, which can be simply quantified and will be discussed in more details in Chapter 4.

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. Perhaps a new algorithm could take in the same input–which sculptures should be connected–and produce a more pleasing drawing which satisfied this architect.

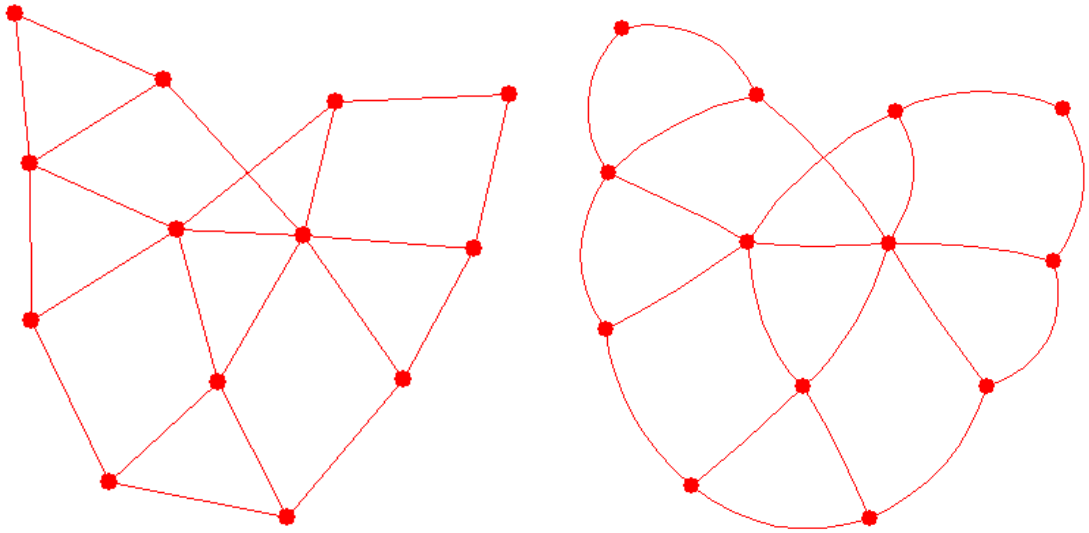

Figure 3.1: Left graph laid out with KK algorithm, right with the curvilinear method using KK

3.3 An Example

Figures 3.1 and 3.2 show how drawing made by the KK [11] and GEM [7] algorithms can be smoothed and aesthetically improved using the method set forth in this paper.

3.4 The Principle of Moving Edges

The theory behind the method is to give edges a mass-like quality so they can also be pushed and pulled and acted upon by forces. In most regular graphs, the vast majority of the visual content is in the edges. Thus, it is logical to use an algorithm that lays out both vertices and edges.

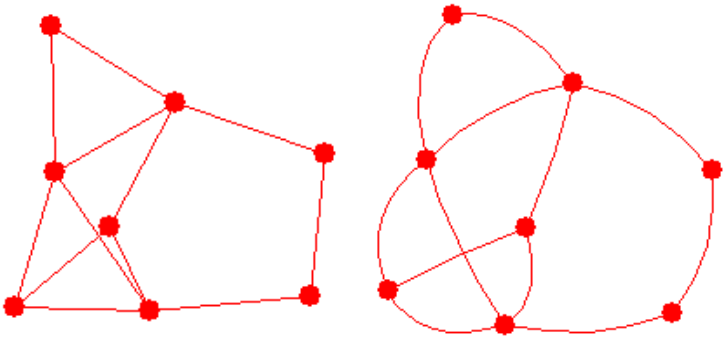

Figure 3.2: Left graph laid out with GEM algorithm, right with the curvilinear method using GEM

Chapter 4

Method for Curvilinear Drawings Using the Force Directed Method

4.1 Measuring a Graph's Aesthetic Properties

There are many measures for the aesthetic qualities of a graph. With an orthogonal drawing, one might be concerned with the number of bends, or with a drawing on a grid one might want to minimize the area of a graph. For the curvilinear graphs used in this paper, we will look at three relevant measures: angular resolution, edge separation, and number of crossings.

Angular resolution measures the angles made by incident edges of a vertex. In general, acute angles are not desirable features. In this paper, angular resolution will be calculated by taking the difference between the smallest actual angle and the optimal angle (360/degree) at a vertex, averaged over each vertex in a graph. Lower numbers equate layouts which are closer to the optimal, and thus, better.

Edge separation is measured by finding the smallest distance between each edge and another non-incident, non-intersecting edge. The sum of this distance for all edges is then normalized by the size of the graph. Greater edge separation means the edges of a graph are more spread out and the graph is more readable.

Number of crossings is very straight forward. We simply measure the number of edge intersections in the graph. In general, crossings are undesirable and should be minimized.

4.2 Reusing the Wheel

There are two basic strategies when attempting to draw a general graph with curved lines and pleasing qualities as previously described: make a new algorithm, or adapt the graph to use a preexisting algorithm. While both methods are viable, the second method is preferable because it can modularly plug into a wide variety of known algorithms, each with its own advantages.

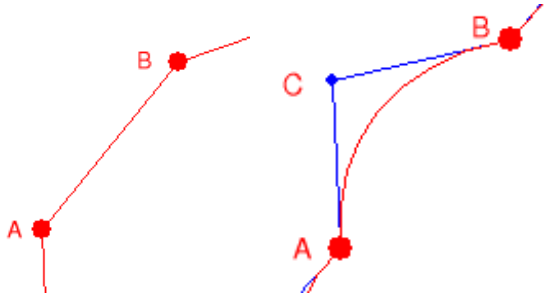

Figure 4.1: On the left, a straight edge; on the right, edge split, and curve drawn with control point

However, an algorithm independent solution cannot use any particular features of an algorithm, so our solution modifies only the vertices and edges of the graph itself, and then appends a different drawing technique.

4.3 Methodology

The methodology for curvilinear graph drawing using force directed layout is actually quite simple. For each edge connecting A and B, create a new vertex at the edge's midpoint, delete the edge, and create two new edges, from A to C and from C to B. Now, every time drawing the edge from A to B (which no longer exists), use C as a control point for a curved line from A to B. Omit edges AC and BC, and vertex C from the drawing. Note the algorithm will not differentiate between A, B, or C, even though we are considering them as different types of vertices.

The effect of this process is to embed a hidden vertex in an edge, thus giving the edge the "weight" that we desired. We can embed more dummy vertices (control points) into an edge to magnify the curving effect. In practice, either one or two vertices will produce good results without bloating the graph.

4.4 Fixing Extra Crossings Through Binding

The danger of adding edges and vertices to a graph is that it will also increase the number of crossings. In theory, one could prevent the addition of any new crossings by imposing boundaries on the control vertices. However, most practical force directed algorithms, while evaluating a modified graph, will sometimes generate new crossings. Luckily, many of these crossings occur in incident edges and can be easily detected.

This is a particular feature of using curved lines drawing; obviously with straight line drawings we can have no incident edges intersecting except at the vertex.

Any number of algorithmic heuristics could be used to unwind these crossings, but binding was designed to be simple and algorithm independent: its only mechanism is adding hidden edges.

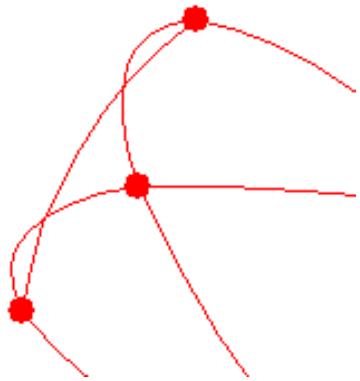

Figure 4.2: Incident edge crossing

To bind a vertex, connect the closest control vertices around that vertex in the order 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 won't 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 the 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. It this way, the bindings can be done in several passes, or all at once, yielding different results.

It should also be noted that if using a more sophisticated layout algorithm, specifically one that minimizes the number of crossings, these local intersections can be avoided, and there should be no need for binding. However, these algorithms are significantly more complex and time intensive, and can be too inefficient for larger graphs.

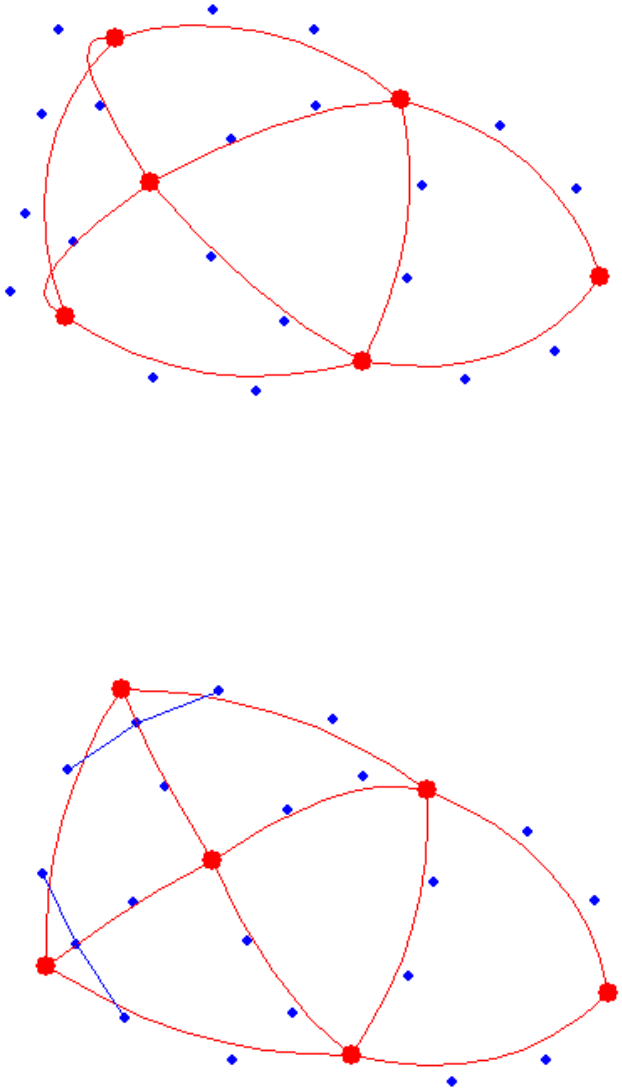

Figure 4.3: A curvilinear graph before and after binding

Chapter 5

Implementation

An implementation of curvilinear graph drawing with the forced directed method was made in Java using the JDSL library from Brown University. It was tested on a series of Linux machines with 1.5 GHz AMD Athlon CPUs and 512MB RAM. The graphs tested were the Rome Graphs from obtained from graphdrawing.org.

The algorithms that were chosen were GEM [7], and KK [11], based on their effectiveness, speed and simplicity . GEM and KK are good choices because they are all around performers [3] and will be an accurate measure of the success of the method. Implementations were adapted from the JDSL library (GEM) and Auburn University (KK).

When drawing the curves, their are any number of splines that could be used. Of course, different situations have different requirements, but in general we want to pick a spline which is smooth and simple. The choice of Bezier curve was made because of its attractive curves and its geometric properties. Its stays within the convex hull of the polygon made by its control points, and its slope starts directly towards the control point. Edges with single and double control points use quadratic and cubic Beziers, respectively. For binding crossed vertices, there are two successive passes. We also test the effect of binding all vertices.

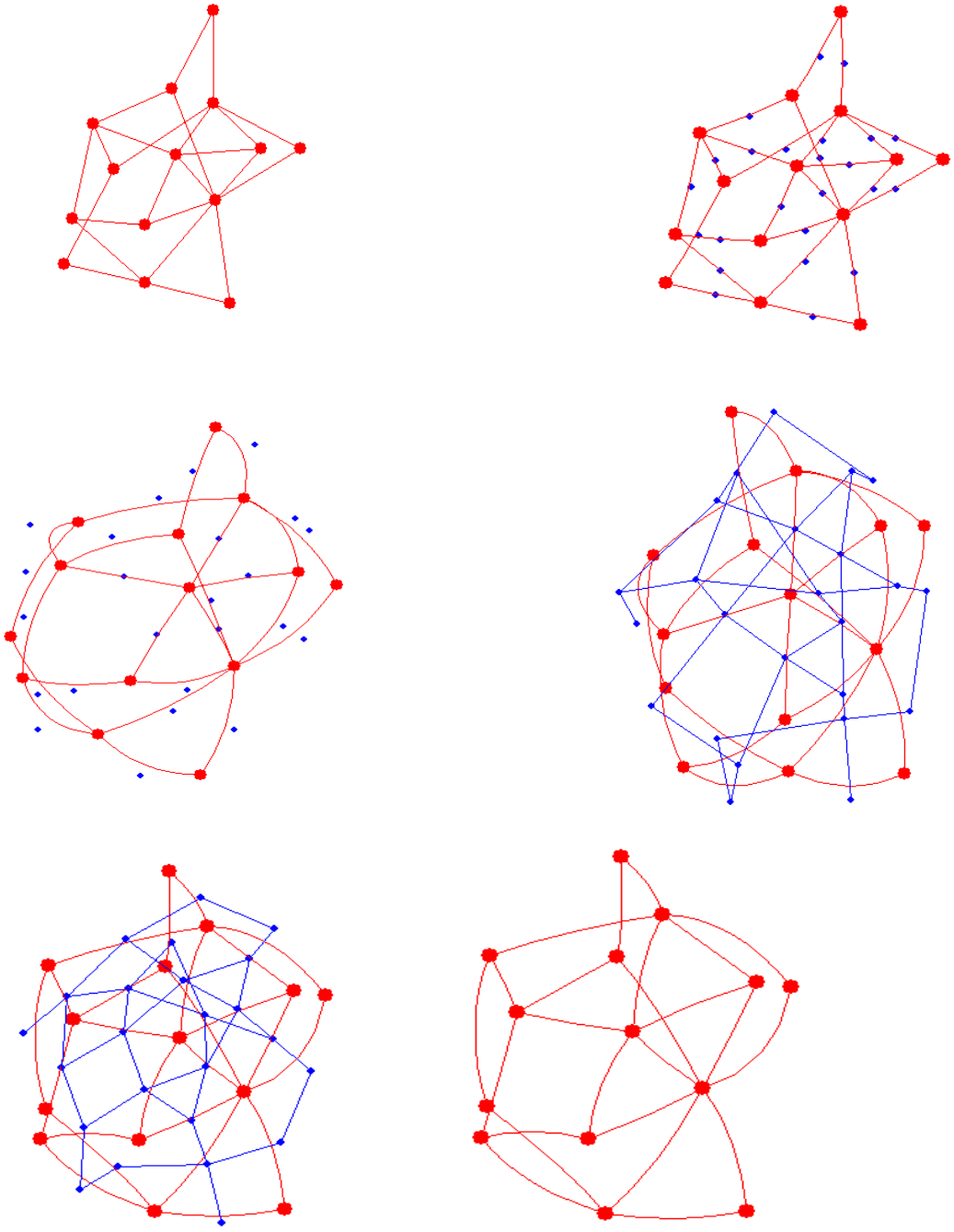

Figure 5.1: An example of the curvilinear method using quadratic Beziers and the "bind all" heuristic

Chapter 6

Experimental Results

The experiments were performed as follows. Each Rome Graph was initially laid out with random locations. Then the algorithm would execute on the graph without the curvilinear method, for comparison. Once completed, the graph's angular resolution, edge separation, and number of crossings were measured as described in Section 4.1.

After the measuring the graph without the curvilinear method, the process was repeated. The layout started with the same random initial placement as before, with the addition of the dummy control vertices evenly spaced on the edges. This process was then done identically 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 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 algorithm ran again and the graph was measured.

edge type, type of binding	angular resolution	edge separation	number of crossings
straight edges (without method)	45.78	.04447	28.14
quadratic Bezier, none	34.28	.04473	30.74
quadratic Bezier, crossed	34.00	.04446	31.08
quadratic Bezier, all	31.58	.04547	30.93
cubic Bezier, none	31.11	.04510	30.26
cubic Bezier, crossed	28.93	.04537	29.47
cubic Bezier, all	24.75	.04613	28.93

Table 6.1: Averages of aesthetic properties using curvilinear method with GEM on the Rome Graphs.

The results of the averages and variances show how the curved lines significantly improve angular resolution, one of the most crucial features of graph readability. With the added heuristic of binding all vertices, using cubic Bezier curves led to an overall average angular resolution 46 percent closer to optimal, with an average increase of less than one crossing per edge. The ranges also reflect the

edge type, type of binding	angular resolution	edge separation	number of crossings
straight edges (without method)	98.53	9.45	1162.87
quadratic Bezier, none	54.27	9.49	1422.42
quadratic Bezier, crossed	56.63	9.14	1445.16
quadratic Bezier, all	58.71	9.66	1448.90
cubic Bezier, none	48.68	9.96	1351.51
cubic Bezier, crossed	40.27	9.83	1301.90
cubic Bezier, all	30.88	9.95	1277.20

Table 6.2: Variances of aesthetic properties using curvilinear method with GEM on the Rome Graphs.

edge type, type of binding	angular resolution	edge separation	number of crossings
straight edges (without method)	9.51 - 82.14	.010 - .197	0 - 222
quadratic Bezier, none	7.54 - 56.08	.011 - .188	0 - 262
quadratic Bezier, crossed	7.07 - 61.06	.005 - .181	0 - 264
quadratic Bezier, all	6.13 - 63.75	.007 - .217	0 - 282
cubic Bezier, none	6.10 - 49.83	.007 - .187	0 - 249
cubic Bezier, crossed	7.18 - 47.03	.005 - .192	0 - 246
cubic Bezier, all	3.65 - 42.52	.011 - .202	0 - 256

Table 6.3: Ranges (min-max) of aesthetic properties using curvilinear method with the GEM on the Rome Graphs.

significant improvement in angular resolution, though they reveal less about the changes in edge separation and number of crossings.

It should be noted that binding in general was not very effective when using quadratic Bezier edges. This is because both vertices share every control point, so when one vertex pulls it to fix a crossing, it often will create a new crossing at the opposite vertex. For cubic Bezier curves, the results show binding is an extremely effective heuristic; it's best when used on all vertices, not just the ones with local intersections.

Unlike the GEM algorithm, the KK did not perform well with the curvilinear method. This method increases the input size by $O(|E|)$, and this has a variable effect, depending on the speed on the algorithm. Because of its $O(n^3)$ running time per iteration compared to GEM's $O(n^2)$, KK's efficiency scaled very poorly and made it prohibitively slow on large graphs. More than that, the KK algorithm uses graph-theoretic distances, and the addition of all the dummy vertices hampers the effectiveness of the algorithm itself, creating far too many new crossings to justify any improvement in angular resolution or edge separation.

Chapter 7

Conclusion and Future Work

The results of experimentation on the Rome Graphs prove that the curvilinear drawing method described here can significantly improve the angular resolution, and thus overall readability, of a graph layout. The mild improvement in edge separation will certainly have a less striking effect on overall aesthetics, but is still good. These benefits must be weighed with the cost of a small increase in the number of crossings. In visual examples, such as maps and charts, this could be a worthwhile tradeoff. But in situations where crossings are most critical to minimize, this method might not be appropriate.

Our experimentation also showed that the curvilinear drawing method will not work successfully with every force directed algorithm. Though it was very effective using GEM, it did not work on larger graphs using KK. So the method should be tested to ensure it will work with a given drawing algorithm.

Future research should investigate the effect of this process with faster methods like that of Gajer and Kobourov [9] and Harel [10], and slower, crossing minimizing algorithms like DH [5], as discussed earlier. It might also be interesting to test the effect of using interpolating splines like the Catmull-Rom, which pass through their control points, instead of Bezier curves.

Finally, designing the method to work with a particular algorithm could allow the ability to add sophisticated features, like preserving edge crossings [2], making it more customizable and effective in a variety of applications.

Bibliography

[1] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[2] Francois Bertault. A force-directed algorithm that preserves edge crossing properties. *Proc. Graph Drawing*, pages 351–358, 1999.

[3] Franz Brandenburg, Michael Himsolt, and Christoph Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. *Proc. Graph Drawing*, pages 76–87, 1995.

[4] Ulrik Brandes and Dorothea Wagner. Using graph layout to visualize train interconnection data. *Proc. Graph Drawing*, pages 44–56, 1998.

[5] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, October 1996.

[6] Peter Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.

[7] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. *Proc. Graph Drawing*, pages 389–403, 1994.

[8] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software, Practice and Experience*, 21(11):1129–1164, 1991.

[9] Pawel Gajer, Michael T. Goodrich, and Stephen G. Kobourov. A fast multi-dimensional algorithm for drawing large graphs. *Proc. Graph Drawing*, pages 211–221, 2000.

[10] D. Harel and Y. Koren. Graph drawing by high-dimensional embedding. *Proc. Graph Drawing*, 2002.

[11] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.