Improved force-directed layouts

Emden R. Gansner and Stephen C. North

AT&T Labs, 180 Park Ave., Florham Park, NJ 07932, USA.
{erg,north}@research.att.com
http://www.research.att.com/info/{erg,north}

Abstract. Techniques for drawing graphs based on force-directed placement and virtual physical models have proven surprisingly successful in producing good layouts of undirected graphs. Aspects of symmetry, structure, clustering and reasonable vertex distribution arise from initial, formless clouds of points. However, when nodes must be labeled and point vertices are replaced by non-point vertices, simple force-directed models produce unreadable drawings, even for a moderate number of nodes. This paper describes the application of two post-processing techniques that can be applied to any initial vertex layout to produce uncluttered layouts with non-point nodes.

1 Introduction

In general, drawing undirected graphs is problematic. The difficulty lies in there being too much freedom. If structure is imposed on the graph, workable techniques arise. For example, if the graph is interpreted to have a directed flow, one can employ the Sugiyama-style algorithms[25, 12]. Alternatively, one can suitably restrict the layout style in order to get a handle on the problem. As an example here, we can consider the class of orthogonal layouts, for which a collection of well-developed and analyzed algorithms is available[26, 27, 8]. But without using such restrictions, there is, at present, no simple non-heuristic algorithm for efficiently drawing general undirected graphs.

The most effective techniques for handling undirected graphs are based on virtual physical models. These techniques, going back to Eades[6] and, computationally, to Kruskal[17], represent the vertices of a graph as physical objects subject to various forces, natural and unnatural. Some subset of the forces encodes the edge information of the graph, typically as an attractive force between the two endpoints of the edge. The object then becomes one of repositioning the nodes in order to minimize the energy of the system or to achieve a stable configuration vis-a-vis the forces acting on the particles. Standard techniques, such as steepest descent or discrete iteration, can be used to search for the desired configuration, although there is always the possibility of only finding a local minimum. Once the final node positions are determined, the drawing is typically completed by connecting edge endpoints with line segments.

In practice, these layout methods are remarkably good, especially given the naive nature of the algorithms (cf. [18, 7]). The resulting drawings typically capture many symmetries of the graph, while avoiding the expensive computations

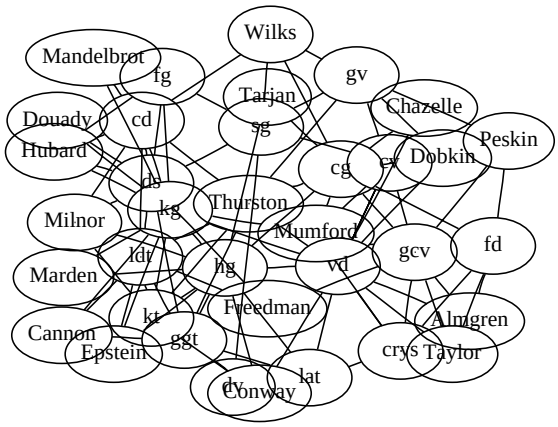

Fig. 1. The effect of non-point nodes

involved in explicitly looking for symmetries[19]. In addition, these methods identify significant clusters of nodes while producing a reasonable distribution of the nodes. There has been a variety of work on this class of algorithms[14, 10, 4, 9, 24, 2, 15, 13, 3], leading to some quite efficient algorithms that can handle "medium-sized" graphs.

Despite the efficacy of these approaches, problems arise when they are applied to the practical case of drawings with non-point nodes. Due to the varying shapes and sizes of the nodes, the resulting drawings tend to be cluttered, with nodes overlapping, if not totally hiding, each other, and edges visually lost in the confusion of being routed over or under the nodes. These problems are particularly prevalent in the common case of graphs with little symmetry and much connectivity. A typical example is shown in Fig. 1. Even modest sized graphs become unreadable. Despite this, such drawings appear in the literature.

A simple solution is to scale the drawing up, while preserving node sizes, until the nodes no longer overlap (cf. Fig. 2). Although this is simple and creates a similar layout, preserving symmetries, it can be wasteful of space, and one must still handle edge-node overlaps. We ask if there is a better way to modify an initial layout, repositioning the nodes to avoid overlap, while preserving proximity, clusters and the overall structure and layout, and using less area. Given that most of the graphs that arise in practice have little symmetry, we are willing to forego preserving symmetries.

This paper describes the use of two general-purpose post-processing mechanisms for improving these layouts to make the drawings more readable using modest additional area. Inspired by work of Lyons et al.[21], the first pass removes overlapping nodes by iteratively constructing a Voronoi diagram for the graph and moving nodes within their Voronoi cells. The second pass, based on

previous edge routing work[5], draws edges as smooth curves to avoid node-edge overlaps.

In the following section of this paper, we present a more detailed description of the post-processing heuristics that we employ. This is followed by a discussion of our current implementation of these heuristics, and sample output and timings. We conclude with a consideration of related work and future avenues for this work.

2 Cleaning the Layout

Given a cluttered layout, such as the one shown in Figure 1, our goal is simple: gradually adjust the nodes until there is adequate space between them, while trying to preserve their relative positions and distances, and then construct the edges as smooth curves avoiding the nodes.

To achieve the first part, we use a variant of a technique developed by Lyons to break up clusters of nodes in a graph drawing and achieve a more uniform distribution. In our version, we construct a window containing the centers of all the vertices, and then construct the Voronoi diagram relative to this window, using the vertex centers as sites. The centroid of the Voronoi cell associated with a vertex represents the point most removed from any other vertex, and it is to that point we move the vertex. By iterating this process, the vertices move away from each other, reducing overlaps. At the same time, this motion largely maintains the relative node positions from the initial layout, as desired. Of course, if the current window is not large enough, each vertex will converge to the centroid of its cell and no further adjustment is possible. At times, therefore, it is necessary to increase the window size and allow the drawing as a whole to expand. Eventually, the repositionings and expansions are sufficient to remove all node overlaps.

Once the nodes have been moved so that there are no overlaps, we then draw the edges. We do not want to use line segments since, even with the additional space, they will conflict with the node representations and obscure the graph structure. Basically, edges should avoid nodes except for endpoints. The use of polylines is possible, but we rejected these for aesthetic and cognitive reasons[1]. We feel that some form of smooth curve works best for edges, reducing to a straight line where possible.

To construct a curve representing an edge, we first find the shortest path connecting the midpoints of the two nodes and avoiding the interiors of any other nodes. This relies on Dijkstra's algorithm and a one-time construction of the visibility graph of the vertices of the polygonal nodes[2]. We then attempt to connect the two nodes by a line segment or a Bezier curve segment. If none of these can avoid crossing into the interior of a node, we find the vertex on

[1] A further discussion of these issues, and our view of them, can be found in Dobkin et al.[5].

[2] The cost of constructing the visibility graph is amortized over the construction of all the edges.

the shortest path furthest from the trial curve, break the problem into two subproblems and solve each recursively. Further details can be found in [5].

3 Implementation

Our current prototype implementation is based on a simple pipeline of two programs, with two elements repeated and using the dot[11] format for describing graphs. The first component of the pipeline is neato[16], which takes an undirected, attributed graph and lays out the vertices as points using an implementation of the Kamada-Kawai[14] algorithm, itself a close relative of Kruskal's multidimensional scaling approach [17, 3]. In addition to determining node coordinates, it also specifies the polygon associated with a node[3].

The second component of the pipeline takes the nodes and their enclosing polygons, and radially expands the polygons by a given amount. This ensures that there is adequate space between the nodes in the final layout. In addition, it determines an initial enclosing window for the vertices, either from user specifications or by expanding the enclosing isothetic rectangle by a given percentage, typically 5%. It then applies the procedure, described in the previous section, of iteratively computing the Voronoi diagram of the vertices and moving each vertex to the centroid of its Voronoi cell, expanding the window when necessary, until none of the expanded polygons overlap. It then emits the graph with the new node positions. We use Fortune's $O(n \log n)$ algorithm to compute Voronoi diagrams. Polygon intersection checking uses a simplified version of the linear algorithm found in O'Rourke[22], preceded by a bounding box check.

Despite the basic simplicity of this phase, there is still room for many variations. For example, with each iteration, one must decide which nodes to move and when to increase the area. Clearly, if the area is increased, the nodes on the perimeter must be moved in order to take advantage of the new area. Our initial strategy was to only move overlapping nodes as long as progress was being made, i.e., the number of overlaps was reduced. If progress was stalled, we increased the area and moved all the nodes. We found that any more subtle strategies along these lines (e.g., follow the same strategy as above but increase the area only if two consecutive iterations show no progress) only increased the number of iterations required and the final area. And, in fact, we found that the fewest iterations and the least area were produced by using the even simpler approach of always moving all nodes and increasing the area whenever the number of overlaps does not go down.

The final component of the pipeline is another instantiation of neato. In this case, it accepts the given node coordinates, without repeating the Kamada-Kawai phase, and moves to the final layout passes of routing edges as smooth curves and generating device-dependent output. As with most of our tools, output formats include PostScript, GIF, and HPGL, as well as simple dot output. The various component algorithms used to construct edges as smooth curves are described in [5], along with their respective complexities.

[3] Non-polygons, such as circles and ellipses, are approximated by many-sided polygons.

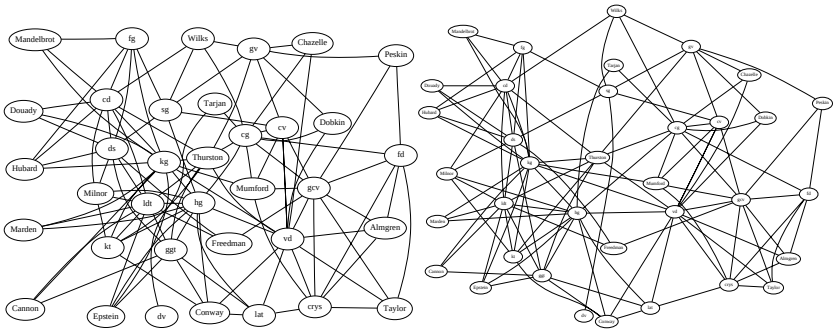

Fig. 2. Tidying the graph

4 Examples

As an example of our approach, we applied it to the graph shown in Fig. 1, and arrived at the drawing given in the left-hand side of Fig. 2. For comparison, the drawing shown on the right-hand side of Fig. 1 was created by dilating Fig. 1, preserving node size, until the nodes no longer touched. Each of these figures is scaled differently to fit the presentation here. If, however, they were scaled equally, so that corresponding nodes on the 3 graphs would have identical size, the drawing on the left in Fig. 2 would be 1.46 times larger than the drawing in Fig. 1, while the drawing on the right would be a factor of 3.9 larger. As for running times, the simple neato layout took 0.18 seconds[4] user time. Our technique required 3.72 seconds, with 0.18 seconds spent in the initial layout, 0.12 seconds used to adjust the node positions, requiring 25 iterations, and the rest used to do the final spline routing. Again for comparison, simply scaling the drawing and then performing the spline routing cost 4.31 seconds.

Table 1 provides additional comparisons between the Voronoi technique and scaling. We applied both methods to 9 graphs that arisen in practice. Columns 4-6 show, for the Voronoi method, the number of iterations needed to remove all node overlaps, the (linear) increase in the bounding box needed for the new layout, and the amount of time this processing took. Columns 7-8 show the size increase and time to remove node overlaps by scaling. We note that the Voronoi approach achieves significantly better use of space. It also takes appreciably longer, but this difference is negligible, as the final, edge routing pass dominates the layout times. The graphs clust.dot, rmf20k50.dot, dpd.dot and houston.jsa.dot are shown in Figs. 3, 4, 2, and 5, respectively.

Additional examples of this technique are shown in Fig. 3 through Fig. 5. In each case, the left layout shows the original straight-line drawing based on Kamada-Kawai; the right layout shows the result after applying our two post-processing adjustments.

[4] All timings reported were done on an SGI Octane.

Table 1. Comparison of applying the Voronoi and scaling techniques for removing node-node overlaps

Graph	Nodes	Edges	Voronoi			Scaling	
			Iterations	Size increase	Time (secs.)	Size increase	Time (secs.)
tube.dot	10	18	3	1.1	0.005	1.49	0.008
clust.dot	16	73	8	1.61	0.045	2.54	0.016
rmf20k50.dot	20	56	2	1.0	0.011	1.49	0.011
inet.dot	24	51	1	1.0	0.012	1.50	0.017
houston.jsa.dot	32	98	2	1.0	0.019	1.64	0.013
dpd.dot	36	108	25	1.95	0.316	3.93	0.031
gte_u.dot	49	260	19	1.77	0.336	3.02	0.044
ngk10_4.dot	50	100	16	1.20	0.294	3.36	0.042
jho5E.dot	213	269	21	1.78	1.843	3.24	0.452

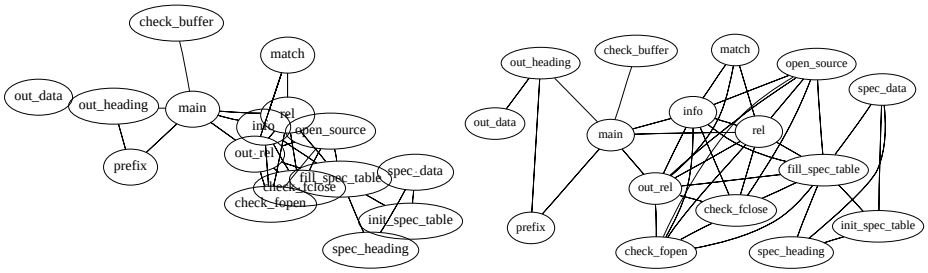

Fig. 3. clust.dot

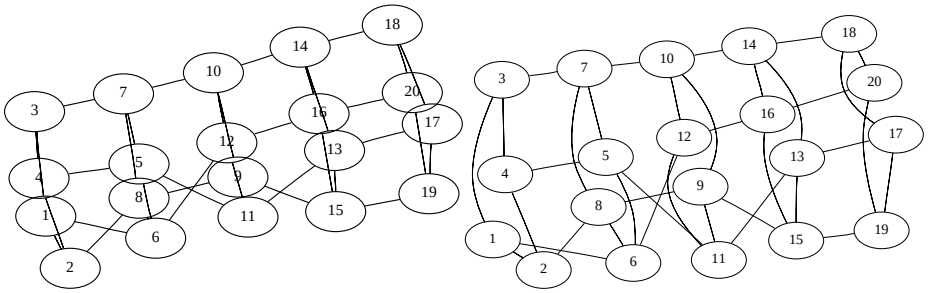

Fig. 4. rmf20k50.dot

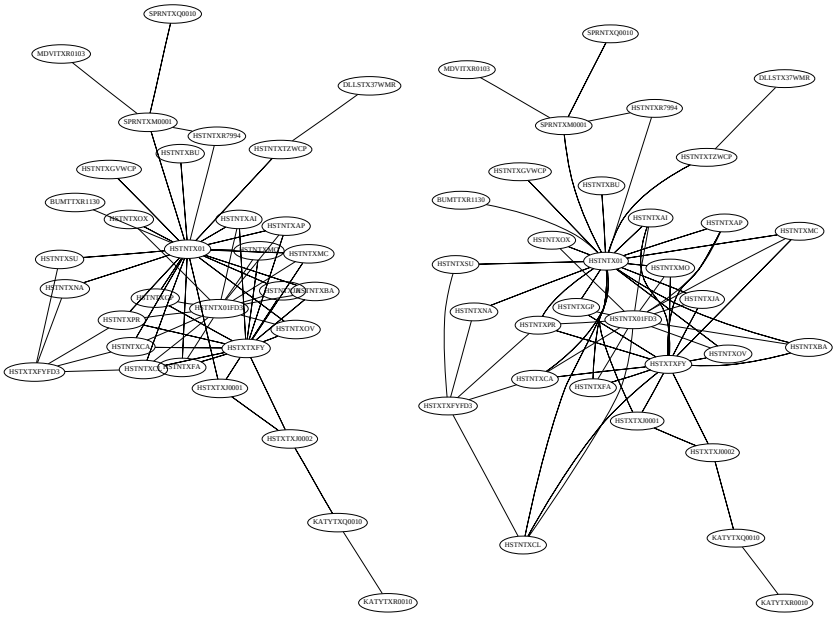

Fig. 5. houston.dot

5 Related Work

Concerning the problem of overlapping objects in straight-line drawings, an alternative approach is to modify the virtual physical model to prevent or at least minimize such overlap. For example, Kamps et al.[15] introduce an additional energy term representing the amount of overlap of the nodes' bounding boxes. Similarly, Wang and Miyamoto [28] modify the forces between nodes, so that if two nodes overlap there is no attractive force between them and there is a stronger repulsive force. Davidson and Harel[4] make nodes and non-incident edges repel.

The problem is that imposing an overlap penalty still is no guarantee that objects are correctly separated. This is particularly apparent in some layouts of large or dense graphs. More research is needed to understand possible interference between the new energy terms and the original model. Besides, the straight-line edge restriction can impose other problems.

Fisheye techniques [23] can adjust spacing between layout objects, but are intended for interactive environments. It is not clear how they can be applied successfully to static layout.

Wills [29] proposes a post-processing phase for virtual physical models. The method is to compute nearest neighbors for all nodes, and increase the separation between node pairs that are too close. It is not clear what happens if there is not enough space locally.

As previously mentioned, our approach was inspired by Lyons et al.[21,20]. They propose improving layouts of points in the plane that typically represent graph nodes. The goal is to more evenly distribute the points within a fixed layout window, while maintaining the basic "shape" of a drawing. The method relies on measures of similarity and distribution. Layouts are updated iteratively by either a Voronoi diagram or spring model, moving points until either the drawing becomes too dissimilar to the input, or a desired level of distribution has been reached. The proposed models for similarity and distribution do not take into account non-point nodes or edges of the graph.

Edge routing can be handled by any technique that satisfies certain aesthetic goals (e.g. make paths that are short, smooth and avoid unnecessary inflections). An alternative to our method described in Section 2 was proposed by Abello and Gansner [1].

6 Conclusions and Future Work

The proposed approach makes readable layouts by a hybrid of virtual physical modeling, geometric constraints and representing edges as smooth curves. The approach is compatible with any initial layout. The resulting diagrams preserve much of the original structure, attempt to use little additional space, and can be constructed efficiently.

This technique also raises some interesting problems. One area concerns the layout itself. The technique prevents unwanted node-node and node-edge intersections, but doesn't address edge-edge intersections. Such intersections can be seen in Figure 2. Global edge routing might be a good way of attacking this problem. The idea would be to determine when edge routes might interact (perhaps when path edges are shared or are proximate) and then to introduce new barriers corresponding to implied routing constraints. Alternatively, it might be possible to introduce some aspects of edge routing into the force-directed model, thereby connecting node and edge layout more closely, as is done in most Sugiyama-style layouts for directed graphs.

Another issue is that the technique of using Voronoi diagrams to break up clusters of nodes really represents a family of algorithms, and leaves open the decisions about what nodes to move, where to move them, when to increase the drawing's bounding polygon, and what type of bounding polygon to use. We discussed some of the variations we tried in Section 3. Adjustment of node positions also potentially interacts with the force model and disturbs local symmetry. More experiments are needed to understand these interactions and good tradeoffs. We notice that our layouts tend to disperse nodes more than necessary.

There is a question concerning whether Voronoi diagrams with polygonal node shapes as sites make better layouts. If so, there is a problem of how to compute these diagrams incrementally as nodes move. Finally, there is a need for robust implementations.

References

1. J. Abello and E. Gansner. Short and smooth polygonal paths. In C. Lucchesi and A. Moura, editors, *LATIN'98: Theoretical Informatics*, volume 1380 of *Lecture Notes in Computer Science*, pages 151–162, 1998.
2. F.J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F.J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 76–87, 1996.
3. J. Cohen. Drawing graphs to convey proximity: an incremental arrangement method. *ACM Transactions on Computer-Human Interaction*, 4(11):197–229, 1987.
4. R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. *ACM Transactions on Graphics*, 15(4):301–331, 1996.
5. D. Dobkin, E. Gansner, E. Koutsofios, and S. North. Implementing a general-purpose edge router. In G. DiBattista, editor, *Symposium on Graph Drawing GD'97*, volume 1353 of *Lecture Notes in Computer Science*, pages 262–271, 1997.
6. P. Eades. A heuristic for graph drawing. *Congressus Numerantium*, 42:149–160, 1984.
7. P. Eades and X. Lin. Spring algorithms and symmetry. In *COCOON'97*, volume 1276 of *Lecture Notes in Computer Science*, pages 202–211, 1997.
8. U. Fossmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F.J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266, 1996.
9. A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I.G. Tollis, editors, *Symposium on Graph Drawing GD'94*, volume 894 of *Lecture Notes in Computer Science*, pages 388–403, 1995.
10. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software – Practice and Experience*, 21(11):1129–1164, 1991. also as Technical Report UIUCDCS-R-90-1609, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1990.
11. E.R. Gansner, E. Koutsofios, S.C. North, and K.P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, March 1993.
12. E.R. Gansner, S.C. North, and K.P. Vo. Dag – a program that draws directed graphs. *Software – Practice and Experience*, 18(11):1047–1062, 1988.
13. W. He and K. Marriott. Constrained graph layout. In S.C. North, editor, *Symposium on Graph Drawing GD'96*, volume 1190 of *Lecture Notes in Computer Science*, pages 217–232, 1997.
14. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31:7–15, 1989.
15. T. Kamps, J. Kleinz, and J. Read. Constraint-based spring-model algorithm for graph layout. In F.J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 349–360, 1996.
16. E. Koutsofios and S. North. Intertool connections. In B. Krishnamurthy, editor, *Practical Reusable UNIX Software*, chapter 11, pages 300–315. John Wiley & Sons, New York, 1995.
17. J. Kruskal and J. Seery. Designing network diagrams. In *Proc. First General Conf. on Social Graphics*, pages 22–50, 1980.
18. X. Lin. *Analysis of Algorithms for Drawing Graphs*. PhD thesis, Department of Computer Science, University of Queensland, 1992.

19. R. Lipton, S. North, and J. Sandberg. A method for drawing graphs. In *Proc. ACM Symp. on Computational Geometry*, pages 153–160, 1985.
20. K. Lyons. Cluster busting in anchored graph drawing. In *Proceedings of the 1992 CAS Conference*, pages 7–16, 1992.
21. K. Lyons, H. Meijer, and D. Rappaport. Algorithms for cluster busting in anchored graph drawing. *Journal of Graph Algorithms and Applications*, 2(1):1–24, 1998.
22. O'Rourke. *Computational Geometry in C.* Cambridge University Press, Cambridge, 1994.
23. M.-A. Storey and H. Muller. Graph layout adjustment strategies. In F.J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 487–99, 1996.
24. K. Sugiyama and K. Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In R. Tamassia and I.G. Tollis, editors, *Symposium on Graph Drawing GD'94*, volume 894 of *Lecture Notes in Computer Science*, pages 364–375, 1995.
25. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(2):109–125, 1981.
26. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Computing*, 16(3):421–444, 1987.
27. R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-18(1):61–79, 1988.
28. X. Wang and I. Miyamoto. Generating customized layouts. In F.J. Brandenburg, editor, *Symposium on Graph Drawing GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 504–515, 1996.
29. G. Wills. Nicheworks - interactive visualization of very large graphs. In G. DiBattista, editor, *Symposium on Graph Drawing GD'97*, volume 1353 of *Lecture Notes in Computer Science*, pages 403–414, 1997.