Interactive Graph Drawing on the World Wide Web

Úlfar Erlingsson   Mukkai Krishnamoorthy
{ulfar,moorthy}@cs.rpi.edu

Abstract

We discuss a system for performing interactive graph drawing on the World Wide Web (WWW), implemented in the Java programming language. The system allows for highly interactive experimentation in graph drawing, supporting direct user interaction and parameter adjustment during the embedding process. The use of Java and the WWW in its implementation makes the system globally available, both for interactive use and for integration into other systems, regardless of computer platform details. We present the design, implementation and use of the system, as well as some experimental results.

1 Introduction

We present a World Wide Web (WWW) based system for interactive experimental graph drawing. The system is written in Java [12], a WWW programming language, and is therefore available for use on almost any computer system connected to the Internet. The system supports a very high degree of user interaction and parameter adjustment, allowing direct interaction with the graph layout during the embedding process. The system was inspired by a small Java demonstration program found on the WWW [19], and motivated by the need for an interactive WWW presentation environment for graph-based information. The system may be used on the Internet from URL `http://www.cs.rpi.edu/projects/pb/graphdraw/` through any Java-aware browser, such as Netscape.

Since the field of graph drawing algorithm design originated with [18] and [13], it has grown tremendously, especially in the last decade as high-resolution graphical devices have become commonly available. A good survey of the field, containing most relevant references, can be found in [2]. However, graph drawing systems, such as the GraphEd system [11], have so far been limited to specific computer platforms and environments. This is an unfortunate obstacle to easy experimentation and usage of graph drawing implementations. Attempts to remedy this based on server-side solutions, e.g., Diagram Server [9] and GraphPack [14], are inherently restricted by network bandwidth and server size. The WWW-based implementation methods presented in this paper show great promise for improving this situation.

There are two main characteristics that distinguish the system presented in this paper from other graph drawing systems: the ability to vary the parameters of the embedding interactively to get an appropriate layout of the given graph, and the fact that the system is inherently distributable and portable, with almost no installation cost across different platforms. In the remainder of this section we will describe the algorithms and methods used in our system. In the next section we discuss the importance of the WWW as a platform for graph drawing. In section 3 we present the implementation of our system, its design and use in experimental graph drawing. Finally, in section 4, we present the results of some experiments in graph drawing using our system.

1.1 The Graph Drawing Algorithms

We focus on the subfield of graph drawing which is concerned with the automatic generation of aesthetically pleasing drawings for humans, as presented in [2], [17] and [10]. In Table 1 we see some defining characteristics of graphs, graph drawing, and graph-drawing algorithms. Graphs can be of many different *types*, be drawn in many different *styles*, with different *aesthetic goals* and different *methods*. We direct our attention primarily to general directed graphs, drawn without constraints, using straight- and polyline edges and rectangular vertices. The methods employed include depth-first search, force-directed placement (or simulated annealing), linear systems solving and various heuristics.

Graphs	**Styles**	**Aesthetics**	**Methods**
Trees	Straight-line	Minimum area	DFS
Planar	Polyline	Balanced	Annealing
DAGs	Spline	Minimum bends	Grid
General	Rectangle-vertex	Minimum crossings	Linear Algebra
Specific	Circle-vertex	Symmetric	Linear Progr.
Bipartite	Constrained	Uniform density	Heuristic
		Orthogonal	
		Minimum edge length	
		Non-overlapping vertices	

Table 1: Some characteristics of graph-drawing algorithms.

Algorithms for drawing directed graphs and DAGs can be found in [8], [7] and [15]. Most of these algorithms draw general directed graphs by initial cycle-breaking by edge reversion and a subsequent DAG algorithm. The DAG algorithms from Bell Labs, described in [8] and [7], use a more complicated multi-pass algorithm which combines depth-first search, linear programming and heuristics. An example of a hierarchical embedding can be seen in the left half of Figure 1.

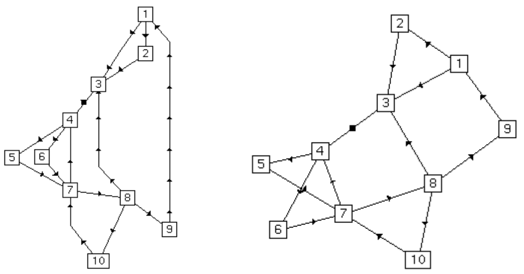

Figure 1: Hierarchical and force-directed embeddings of a control flow graph.

Drawing of general graphs is arguably best done using the force-directed methods of [16], [5] and [6], which are derivatives of Eades' early work in [3]. These methods use a simulated model of the graph in a pseudo-physical setting, replacing the edges with (magnetized) springs and the vertices with repulsive magnets. The graph is then laid out using an iterative simulated-annealing process, where the minimum-energy layout approximates the most aesthetically pleasing one. These methods are good for drawing balanced, uniformly distributed graphs with similar edge lengths, and the implementation of [16] can minimize edge crossings. The right half of Figure 1 contains an example of this embedding.

2 Graph Drawing on the World Wide Web

Graph drawing on the World Wide Web (WWW) has several potential advantages over conventional graph drawing systems. A good WWW-based graph drawing system can be used to display graph-based information by any networked computer in the world, without requiring any special installation, and regardless of the specifics of the particular computing platform and graphical environment. WWW-based graph drawing is therefore both practical and useful, and has the potential of making both experimental and applied graph drawing universally available.

As far as the authors know, there were only two graph drawing systems available on the WWW at the time this paper was written. One was the experimental GraphPack [14], based on cgi-bin scripts, and the other one was the Java graph drawing demonstration program from Sun [19], which motivated this paper. The server-side solutions of GraphPack and Diagram Server [9] suffer from several disadvantages, namely high network overhead, possible server overload and a somewhat complicated network-specific installation. The Java program from Sun was written as a demonstration of the capabilities of the Java language and is therefore understandably severely limited in its features.

The system described in this paper is written in Java and the JavaScript scripting language, in such a way that all computation takes place at the client side, i.e., on the user's computer. This eliminates the need to make slow network connections in drawing each graph, and allows the graph drawing to be completely interactive and dynamic.

Our system supports many different kinds of embeddings, each implemented to support dynamic modifications and adjustments from the user, even while the embedding is still taking place. Thus the user can, at almost any time, adjust the position of vertices, modify the parameters controlling the embedding, and rotate the graph in three dimensions. This degree of interactivity is uncommon in graph drawing systems, and could not have been realized on the WWW using server-side techniques.

Another benefit of our system, stemming from its WWW-based implementation, is the possibility of integrating it effortlessly into another system, such as an information browser. Another package can use our graph drawing system to graphically present the graph structure of information, with the possibility of embedded hyperlinks in the vertices. This usage does *not* require the encompassing package to install our system, and the end user need not even be aware that our system is not part of the package. Thus, our graph drawing system could be utilized in a succinct and transparent manner to browse objects in a different visualization system.

Our system suffers from the current disadvantages of client-side WWW-based programs, namely that the compiler/interpreter/network is in many cases too slow for practical use. Currently, our system may be sped up by downloading it on the local system, thereby removing the network connection requirement. In the future, these disadvantages will hopefully disappear, as better caching strategies, optimizing compilers and run-time compilation to native code are implemented for WWW-based programs.

3 The Graph Drawing System

The graph drawing system presented in this paper originated as a proof-of-concept prototype, where the Java language was chosen for its capacity for integration into other WWW-based systems, and to allow rapid development. As a prototype, the system was designed from the start to support a high degree of user interactivity, and in a modular fashion so that new embeddings could easily be added. As the prototype was developed, the advantages of WWW-based graph drawing became readily apparent, and the prototype was fleshed out into a more fully featured system.

The system is implemented using the object-oriented metaphor, with separate classes for vertices, edges and embeddings, as well as a central "blackboard" containing global information, such as the set of currently visible nodes. The blackboard contains a set of embeddings, each of which implements the *interface*[1] common to all embedders, i.e., the functions *Initialize* and *Embed*. Addition of a new embedding requires only the implementation of a class satisfying

[1]This is a feature of the Java language adopted from Objective-C as a substitute for multiple inheritance.

the embedder interface and its registration into the blackboard.

Our system is currently started through a set of HTML pages written using JavaScript, which can be run through the Netscape WWW browser. It is easy for the user to input a new graph, for example, if the graph is not overly large, by entering it into an HTML form interface, or, if the graph is to be used more often than once, by creating a special HTML page containing the graph data. If the user employs the proper Java environment it is also possible to store and retrieve the graphs on the local file system, but this is by default disallowed for security reasons.

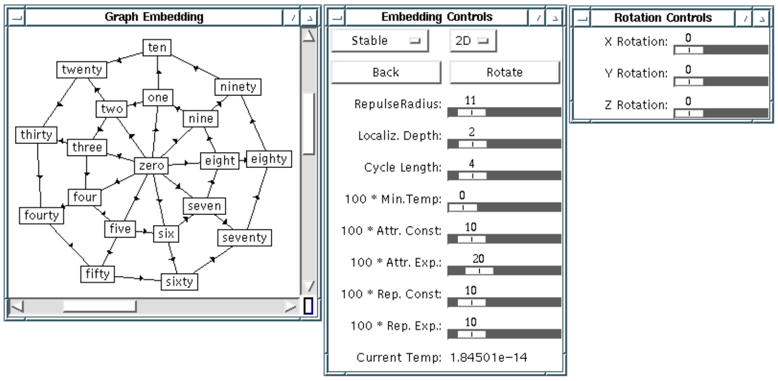

Figure 2: The WWW-based interface to our graph drawing system.

Once the system has been started, its interface consists of three windows: a main drawing panel displaying the graph, with scrollbars to allow panning; a general control window, containing controls allowing for the adjustment of various parameters, such as the embedding used; and a rotation control window. The windows of the interface can be seen in Figure 2. In the rest of this section we list the embeddings and the possible adjustments.

3.1 The Embeddings

The user can at any time select among a number of different embeddings. Most of these reposition the vertices completely, but the force-directed methods iteratively lay out the vertices from their starting position. The methods are as follows:

Random The vertices are placed at random points in the drawing space.

Circular The vertices are distributed evenly on the circumference of a circle.

Force-directed This is an adaptation of Fruchterman and Reingold's simulated-annealing-based algorithm from [6]. Vertices are set to repel each other, while edges attract adjacent vertices. The graph is then embedded using an iterative process.

Hierarchical This is an adaptation of the directed graph methods in [15] and [4]. Cycles are broken by edge reversal, vertices are assigned to levels by their topological ordering, and the resulting hierarchy is iteratively "combed" to minimize edge crossings. This combing is done by repeated reordering of levels using vertex barycenters.

Barycentric This is a variant of Tutte's pioneering method from [18], previously implemented in [14]. Vertex positions are calculated from a set of fixed vertices, e.g., a cycle from the graph laid out in a circle, by the solution of a linear system.

Relaxed This is a force-directed method based on the implementation in [19], with repelling vertices but fixed-length edges.

All the above embeddings are implemented in both two and three dimensions, although only the force-directed algorithms produce very aesthetically pleasing graph layouts in three dimensions. In other respects, the embeddings are implemented for the most part as described in their respective sources.

An exception to this is the level-placement phase of the hierarchical embedding. In [15], the authors lament that their level-placement method does not minimize the total edge length of the resulting graph, and no solution to this problem is mentioned in [4]. Our implementation achieves minimum total edge length by a very simple method. After the initial level placement by topological ordering, we scan the vertices again, now in *reverse* topological order, and *hoist* the vertices to their maximum possible level, i.e., one less than the minimum level at which they are used. Since we do this in reverse topological order, all vertices are hoisted to their maximum possible level, thereby minimizing edge length.

The *relaxed* embedding has not been discussed in the literature, but is a simple modification of the force-directed methods. This embedding is of some interest, however, since it provides an excellent framework for user interaction with the graph. As the user moves vertices around during a relaxed embedding, the active forces provide tactile elastic feedback, which in a sense adds a new dimension to the graph layout, that of dynamic change and interrelation. This enhancement is in many ways analogous to the improved perception of pseudo-three-dimensional scenes when slight random movement is added.

3.2 Use in Experiments

The user of our graph drawing system can interact with the graph embedding process in a variety of ways. Some adjustments, such as explicit movement of vertices, can be made at any point, whereas others are specific to a particular embedding. The user may at any time do any of the following:

- Stabilize the layout, freezing the current position of vertices.
- Select and move vertices around with a "click-and-drag."

- Fix vertices in place, so they are not affected by the ongoing embedding.
- Localize the graph to display only vertices within distance d from a particular vertex.
- Combine a set of vertices into one vertex, either the set of vertices which are fixed in place, or all vertices within distance d of a particular vertex.
- Undo a localization or grouping as described above.
- Change the area of the graph layout drawing space.
- Select whether the embedding is two- or three-dimensional.
- Rotate the graph around any of the tree axes.

Below is a list of the embeddings, with a description of what embedding-specific adjustments the user may perform interactively during their execution.

Force-directed In [6], the original paper describing this embedding, the authors define functions for the attractive and repulsive forces based on the distance between vertices, $f_a(d) = d^2/k$ for attraction, and $f_r(d) = -k^2/d$ for repulsion, where k depends on the square root of the area. In our implementation, these functions are, respectively, $f_a(d) = c_a d^{e_a}/k$ and $f_r(d) = -c_r k^2/d^{e_r}$, where c_a, e_a, c_r and e_r are constants modifiable by the user. In addition the user may select the minimum amount of energy in the annealing process, effectively "turning up the heat" on the placement.

Hierarchical Here, the user may interactively move vertices during the edge-crossing minimization phase, thereby changing the relative ordering of the vertices and assisting the layout process.

Barycentric In this embedding, the user may interactively select a cycle-length, and the graph will immediately be drawn relative to a circularly laid out cycle of that length. Alternatively, the user may fix a set of vertices in place and draw the graph relative to those fixed vertices.

Relaxed Here, the user may modify the fixed length of edges between vertices.

4 Some Experimental Results

In this section, we provide embeddings of the five platonic solids, the Petersen graph, a hypercube, example dependency graphs obtained from makefiles, and control flow graphs arising in compiler optimizations, all interactively laid out with our system.

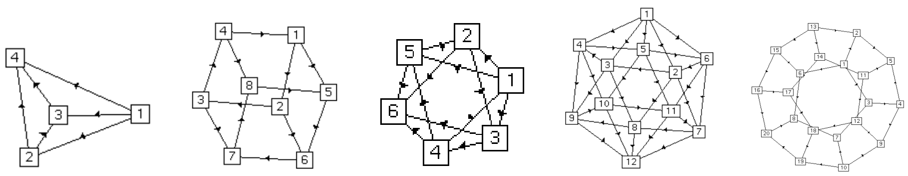

Figure 3: Force-directed embeddings of the platonic solids.

4.1 Platonic Solids, Petersen Graph and Hypercube

The five solids can be seen in Figure 3, and are, respectively, the tetrahedron, the cube, the octahedron, the icosahedron and the dodecahedron. These are very regular graphs and easily drawn using force-directed placement, as they are in the figure.

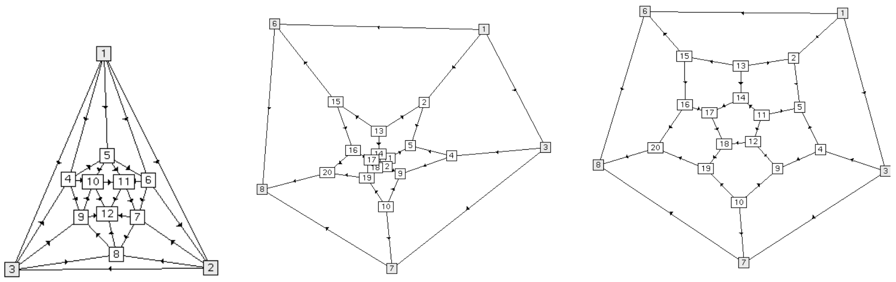

Figure 4: Mostly barycentric embeddings of the icosahedron and dodecahedron.

In Figure 4 we have also included a barycentric embedding of the icosahedron and dodecahedron, using cycles of length 3 and 5 respectively. In the case of the two leftmost embeddings, the outer cycle is fixed and the barycentric embedding has solved for the position of the remaining vertices. In the rightmost graph, however, we have used force-directed placement from the initial embedding in the center of Figure 4. This combination of methods gives excellent results, and is a good example of the power of interactive graph drawing. As we have emphasized earlier, the user can interact with the barycentric embedding in several different ways, e.g., by changing the length of the outer cycle and rotating the graph in three dimensions, the latter of which is in this case surprisingly helpful for visualization.

Figure 5 shows our embeddings of the Petersen graph. Reading from left to right the embeddings are force-directed, barycentric, barycentric followed by force-directed, and, finally, grouped force-directed. The Petersen graph is in general difficult to embed. It does not embed nicely with force-directed placement, as can be seen in Figure 5, whereas the barycentric method gives excellent results, especially when followed by force-directed placement. In the last figure, we have grouped all vertices within unit distance from vertex 7, surprisingly resulting in a hexagon.

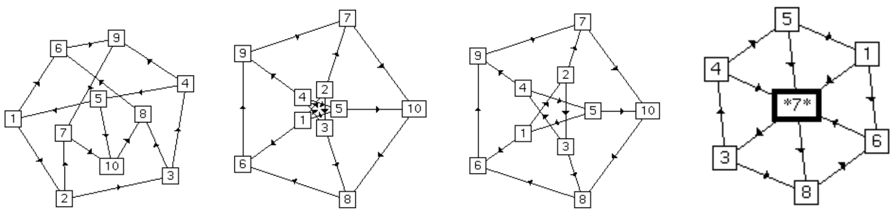

Figure 5: Force-directed and barycentric embeddings of the Petersen graph.

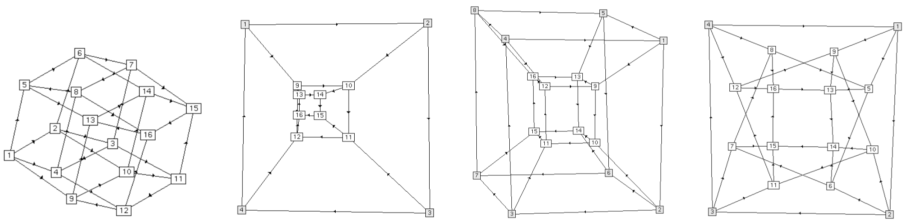

Figure 6: Force-directed and barycentric embeddings of the 4-D hypercube.

Finally, we have also embedded a four-dimensional hypercube. Figure 6 shows our results. The leftmost embedding is the result of a force-directed placement and, while aesthetically pleasing, does not clarify the interrelations of the graph. The second embedding from the left is a barycentric embedding, done with four fixed outer vertices. In this embedding, there are four vertices which are obscured by other vertices. If we fix these obscured vertices in a larger outer rectangle and perform a second barycentric embedding, we get the third embedding from the left. This is the embedding most commonly used to depict the 4-D hypercube. In the rightmost and final embedding, we have combined the barycentric and force-directed methods as before, with good results.

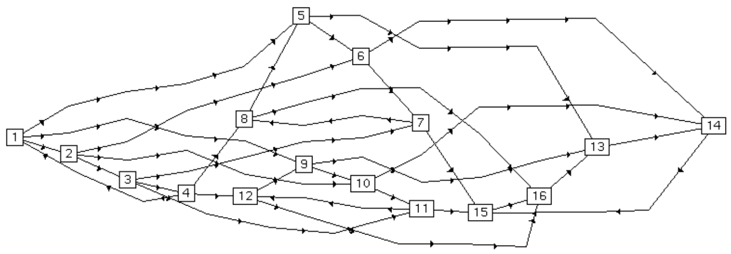

Figure 7: A hierarchical embedding of the 4-D hypercube.

In Figure 7 we can see the outcome of running the hierarchical embedding algorithm on the hypercube. This embedding is of some interest, since, despite its complex nature, it has almost the minimum possible number of edge crossings.

4.2 Makefiles

In software systems, "makefiles" are often used to specify the acyclic dependencies between files and to control compilation. In Figure 8 we can see a hierarchical embedding of the directed acyclic graph from such a makefile.

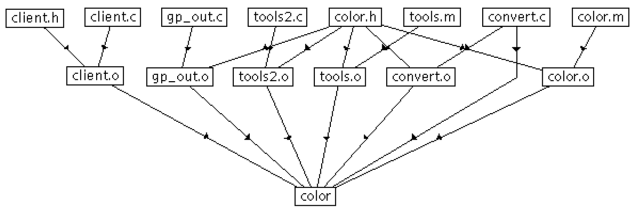

Figure 8: A hierarchical embedding of a simple "makefile" dependency graph.

When used to control large systems, makefiles, and hence their embeddings, can become quite complicated. In Figure 9 we can see the result of simplifying such a complicated makefile by grouping vertices within distance two of the unique highest-level vertex, followed by a force-directed embedding. The resulting figure is not too complicated, and gives a clear idea of which vertices are the leaves.

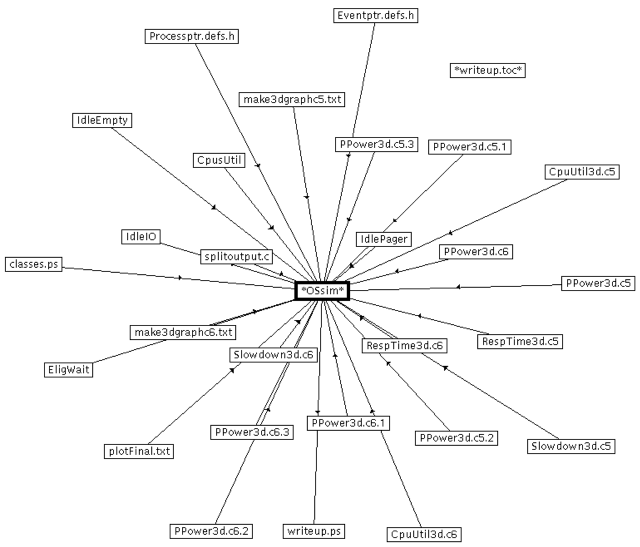

Figure 9: A simplified force-directed embedding of a very large "Makefile".

4.3 Control Flow Graphs

We also experimented with embedding two graphs from earlier publications. In Figure 1 of section 1.1, we showed hierarchical and force-directed embeddings of a control flow graph from Aho, Sethi and Ullman [1], (page 661, Fig. 10.45).

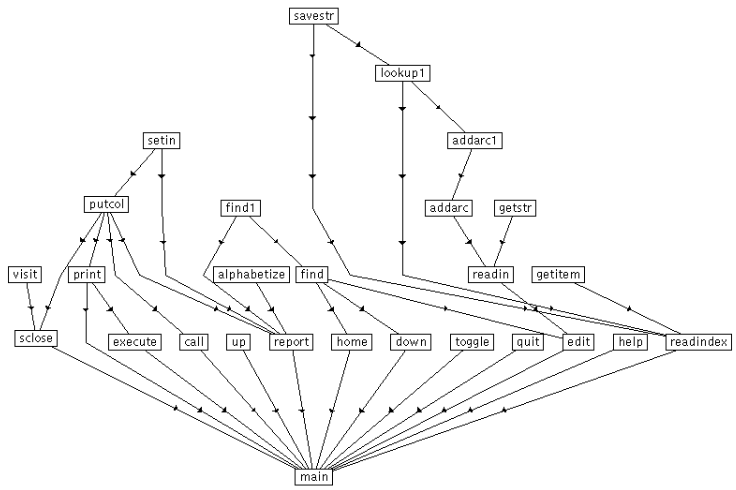

Figure 10: A hierarchical embedding of a function-call graph.

Figure 10 shows a hierarchical embedding of the running example from Rowe et al [15]. During and after the embedding process, we can interactively assist the algorithm in eliminating edge crossing and in the layout of the graph.

5 Conclusions

In our experiments, we found that using the interactive control mechanisms provided in our implementation helped us achieve a clearer understanding and visualization of the graphs we were embedding. Having implemented our system as a WWW-based application, we can easily integrate this control and visualization in other hypermedia applications. Interactivity and easy accessibility are, in our view, two very important issues in the design of any graph drawing system. The WWW-based methods used in the implementation of our system allow for the easy resolution of these issues, and are potentially of great significance for the field of graph drawing.

The reader is encouraged to access and use our system on the Internet through the URL `http://www.cs.rpi.edu/projects/pb/graphdraw/`.

Acknowledgments Parts of our system were implemented as coursework for William Randolph Franklin's course Computational Geometry at Rensselaer Polytechnic Institute. Thanks go to Kendra Willson for proofreading this paper.

References

[1] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1988.

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: An annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.

[3] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[4] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, pages 424–437, 1991.

[5] A. Frick, A. Ludwig, and H. Mehldau. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 388–403. Springer-Verlag, 1995.

[6] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

[7] 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.

[8] 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.

[9] G. Di Battista A. Giammarco, G. Santucci, and R. Tamassia. The architecture of Diagram Server. In *Proc. IEEE Workshop on Visual Languages (VL'90)*, pages 60–65, 1990.

[10] M. Himsolt. Comparing and evaluating layout algorithms within GraphEd. Manuscript, Fakultat fur Mathematik und Informatik, Univ. Passau, 1994.

[11] M. Himsolt. GraphEd: A graphical platform for the implementation of graph algorithms. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 182–193. Springer-Verlag, 1995.

[12] JavaSoft. The java language specification version 1.0 beta. Technical report, Sun Microsystems, 1995.

[13] D. E. Knuth. Computer drawn flowcharts. *Commun. ACM*, 6, 1963.

[14] M. S. Krishnamoorthy, F. Oxaal, U. Dogrusoz, D. Pape, A. Robayo, R. Koyanagi, Y. Hsu, D. Hollinger, and A. Hashmi. GraphPack: Design and features. *World Scientific: Scientific Visualization*, 1996.

[15] L. A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. *Softw. – Pract. Exp.*, 17(1):61–76, 1987.

[16] K. Sugiyama and K. Misue. A simple and unified method for drawing graphs: Magnetic-spring algorithm. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 364–375. Springer-Verlag, 1995.

[17] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst. Man Cybern.*, SMC-18(1):61–79, 1988.

[18] W. T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, 3(13), 1963.

[19] A. van Hoff, S. Shaio, and O. Starbuck. *Hooked on Java.* Addison-Wesley, 1995.