Comparing and Evaluating Layout Algorithms within GraphEd

Michael Himsolt
Fakultät für Informatik und Mathematik
Universität Passau, 94030 Passau, Germany

Graph drawing and layout algorithms gain growing interest and importance for the visualization of complex data structures. But, despite many algorithms and concepts, there is no satisfactory solution to the central problem on the criteria for good and readable layouts. We approach this problem and evaluate layout algorithms by comparing their effect on a large set of sample graphs. In each run we compute statistical data which is collected and evaluated. Our experiments show that traditional layout criteria, such as minimal area or maximal edge length or straight line edges, are not as important as they may appear. Balance is often better than optimization, and displaying the intended and inherited structure of a graph is often more important than these formal cost criteria.

1 Introduction

With the general availability of graphical user interfaces in the early 80's, there has been a growing interest in visualizing complex structures. For this we need both an abstract structure and algorithms that draw the structure automatically. For the first, graphs are well accepted to serve as the abstract structure. A graph G is a pair (V, E) where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. Graphs are used to model different structures such as finite state automata, state charts, flowcharts, database schemata, electrical and VLSI circuits, Petri nets, neural nets, or molecules, to name just a few.

Automatic graph drawing [7] is a current field of research. For specific graph drawing algorithms, graphs must be classified according to their characteristics. However, there is a wide range of such characteristics, such as directed or undirected, with or without cycles, planar or non planar, bounded degree, etc. . Secondly, the application behind the graph has its own structure. The purpose and meaning of a Petri net, for example, is different from the graph theoretic properties of the net. Good graph drawings should reflect both. Moreover, human's evaluate drawings according to further criteria. They may be based on intuition and are not completely known. This point has been stressed, e.g. in [29]

A good graph drawing algorithm should be based on a mixture of all these criteria. It should reflect the properties of the graph, the application structure and the cognitive structure, reflecting humans understanding of a nice drawing.

However, it is not known how all these requirements should be combined for a nice drawing of a graph. We approach this problem by comparing the effect of several different graph drawing algorithms on large sets of sample graphs. We collect geometric properties of the drawings, evaluate them statistically and relate them to the subjective human judgements. From that, we conclude that the ranking of the classical criteria for graph drawing algorithms should be reconsidered and that the algorithms need adaption and revision.

This paper is organized as follows. In section 2, we present an overview on the GraphEd system and the implemented graph drawing algorithms. Section 3 explains our evaluation experiments, and Section 4 shows our results. In Section 5 we give a subjective ranking of layout criteria.

2 GraphEd

GraphEd ([24], [25], [20]) is an extensible editor for graphs and graph grammars. The GraphEd system consists of the following three parts: the core system, the application interface, and the application modules. It it written in C and runs on SUN workstations.

The core system is the graph editor itself. GraphEd's object oriented user interface supports all functions that are necessary for a convenient manipulation of graphs. The fonts and shapes of nodes and styles of edges can easily be customized. As a special feature, graph grammars can be used as macro systems to generate graphs with a specific hierarchical structure.

The application interface is based on Sgraph [23], a data structure for programming with graphs. Application modules have full access to manipulate the graphical representation of nodes and edges. They may create their own windows in the user interface. Algorithms in GraphEd are programmed with Sgraph. The application modules implement complex editor functions, graph theoretic algorithms, layout algorithms, graph grammar algorithms and application specific extensions.

With its capabilities to create and edit graphs, GraphEd provides an efficient environment to create and test large sets of examples. Since all drawing algorithms

are built into one tool, we can compare the effect of different layout algorithms on a graph within the same environment.

GraphEd's graph drawing algorithms are classified into four categories; general graphs (Section 2.1), directed acyclic graphs (Section 2.2), planar graphs (Section 2.3) and graphs with a specific structure (Section 2.4). This agrees with the classification schema in [7], where the reader may find further motivation, criteria and references to other algorithms.

We have chosen and implemented a particular subset of the known algorithms in order to experiment with at least one algorithm from each major class of graph drawing algorithms, and to perform the comparison on algorithms that are significantly different from each other. This shall help in focussing on the essentials.

2.1 Algorithms for general graphs

Our Force Directed Algorithms (FD-K, FD-FR) are based on the algorithms by Kamada and Kawai [28] and by Fruchterman and Reingold [18], which are based on the original algorithm by Eades [11].

Such algorithms impose certain forces upon the nodes of a graph. A heuristic is used to bring them into equilibrium. They produce very good layouts for most graphs, and display isomorphic and symmetric substructures. A major drawback is the high runtime (for a classification of runtimes, see Section 4).

In our implementation, the FD-K approach produces smoother drawings than the FD-FR approach, but it has a noticeably higher runtime.

2.2 Algorithms for directed acyclic graphs

Directed Acyclic Graph Drawing (DAG) is based on algorithms by Sugiyama, Tagawa and Toda [34] and Eades and Sugiyama [13].

This algorithm provides a good base to draw directed acyclic graphs according to their topological sorting. They can also be be applied to general graphs, by temporarily reversing some edges to break up cycles as described in [13]. Our implementation of the algorithm runs at medium speed.

2.3 Algorithms for planar graphs

Planarity is a typical characteristic for graph drawing. There are several algorithms around which draw planar graphs with a planar embedding. For example, standard planarity tests (see e.g. [14]) can be modified to produce a planar embedding, which is a cyclic ordering of the edges around each node. This ordering does not directly define coordinates, but determines a general frame for the layout of a graph. For the drawing algorithm itself, it remains to bring this frame into a nice form.

This has been realized by the following four algorithms:

Planar Orthogonal Grid Drawing with Bends Minimization (POGB) is based on an algorithm by Tamassia [35]. All nodes are placed on a grid, and edges are polylines with only horizontal and vertical segments. The number of bends is minimized with respect to the given planar embedding. This algorithm gives our best layouts, although we must pay for it with the highest running time in this class.

Planar Grid Drawing (PG) is based on an algorithm by Woods [39]. This algorithm gives suitable results for small graphs, but the number of bends is much higher than the one obtained with the POGB algorithm. Theoretically, it has a quadratic bound on the number of bends. Also, the edges may have

arbitrary slopes, which is less pleasing than in the POGB algorithm, where edges are composed of horizontal or vertical segments only.

Note that there are further linear-time planar polyline drawing algorithms that guarantee a linear number of bends, e.g. [1], [10], [37], but these have not yet been implemented in our system.

Planar Convex Faces Drawing with Straight Line Edges (PCS) is based on an algorithm by Chiba, Onoguchi and Nishizeki [3]. Some graphs are drawn nicely with this algorithm. However, it generally tends to cluster nodes, which makes the drawings less readable than those produced by the POGB and PG algorithms. The outer face is always circular. But, the algorithm runs very fast.

Planar Grid Drawing With Straight Line Edges (PGS) is based on an algorithm by de Fraysseix, Pach and Pollack [17] with speed improvements as described by Chrobak and Payne [4]. A similar algorithm has been described by Schnyder [32].

Similar to the previous one, this algorithm tends to cluster nodes. The graph always has a triangular shape. From the subjective quality of its drawings, this algorithm is our worst, although it has the best theoretical characteristics with planar straight line drawings on a grid and quadratic area.

In the literature (see [7]) there are further algorithms for planar graphs which have not been implemented in our system. This is because they are based on PG, POGB and PGS, and the drawings would not be much different.

Moreover, there are some approaches using algorithms for planar graphs for drawing of general graphs after an initial planarisation step ([8], [27], [30]). However, the influence of the planarization step is difficult to evaluate.

2.4 Algorithms for graphs with a special structure

The above algorithms are based on the graph theoretic properties of a graph. They have no knowledge of the structure or the meaning of a graph. A human designer may provide this information, but the lack of it is often the reason for a gap in the readability of the automatic drawings and the man-made drawings. The next algorithms are tailored to graphs with a special structure, and can reduce this gap.

Tree Drawing (T) is based on an algorithm by Walker [38], which is a generalization of the algorithm by Reingold and Tilford [31]. From the practical point of view, tree drawing seems to be the closest to a general solution. Our algorithm reproduces the tree structure in nearly the same way as the user would do it by hand.

Petri Nets (PETRI) Seisenberger [33] has implemented a Petri net drawing algorithm in GraphEd.

The Petri net algorithm takes agents as input, which are term descriptions for special Petri net structures. Graph theoretically, they are similar to series parallel graphs. The drawings are nice, since they draw a Petri net in a way similar to that of a human designer. This comes from the fact that the agents provide detailed information on the structure of the nets, which is exploited for the drawing. However, not all Petri nets have such a term description.

Data Flow Graphs by Graph Grammar (GG-Df) In a case study, we have designed a graph grammar that generates data flow graphs [2].

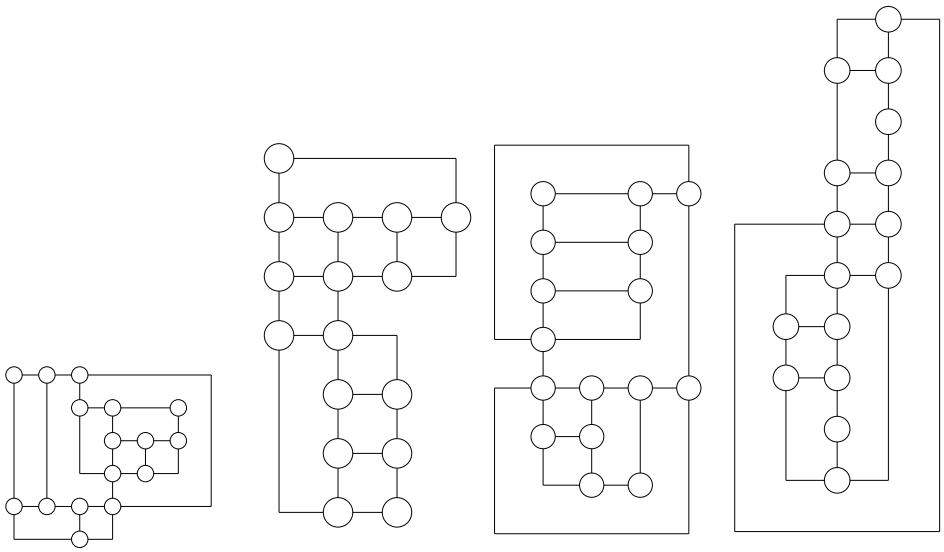

Figure 1: The effect of different planar embeddings in the POGB algorithm.

Graph Grammar Based Algorithms (GG-H) Several variants of layout algorithms based on graph grammars have been implemented by Hickl [22].

These algorithms use a graph grammar for more information on the graph's structure. Graph grammars are natural extensions of string grammars to graphs ([15], [16]). They enrich the graph by a hierarchy. In our implementation, either a parser generates a derivation tree, or the user applies derivation rules interactively.

On the positive side, the structure of a graph can be identified and reflected in the drawing if a suitable graph grammar is used. On the negative side are the parser's high runtime and the need for a suitable graph grammar. Not all graph classes can be described by a graph grammar.

2.5 Summary

Tables 2 and 3 show various graphs drawn with the general and the planar drawing algorithms. We have restricted the examples to planar graphs, since nonplanar graphs can only be drawn with the FD-K, FD-FR and DAG algorithms. From our experience, drawings of nonplanar graphs have a similar appearance, except that there are more crossings. This stimulates an initial planarization step.

Table 1 gives an overview of the layout algorithms which are currently implemented in GraphEd. The term *k-connected* means k-connected for undirected graphs and weakly k-connected for directed graphs (for graph theoretic definitions, see e.g. [14]). *Straight line*, *polyline* and *bends* describe the style of the edges.

Our personal *rating* schema uses one to five stars for the quality of the drawing, where '$\star$' is the lowest and '$\star\star\star\star\star$' is the highest rating. It is purely based on viewing the drawings. We have incorporated judgements made by more than twenty researchers and graduate students in Computer Science at the University of Passau, who have been working frequently with GraphEd.

From our experiments, we have learned that external factors can have a large influence on the graph. As an example, different planar embeddings produce different drawings in Figure 1. More complex effects are responsible for the variations in Figure 2. In this case the quality of the drawing varies over a wide range.

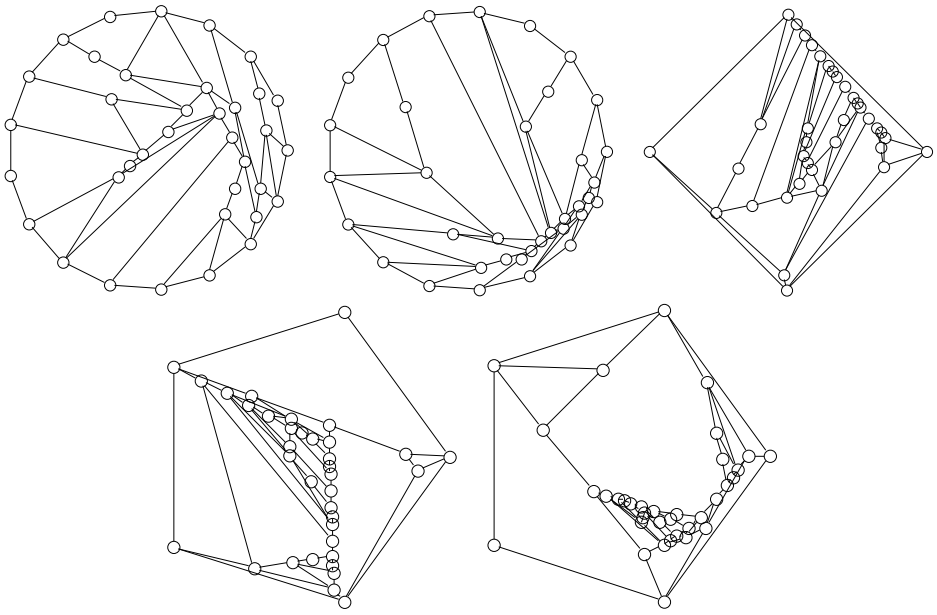

Figure 2: The effect of different planar embeddings in the PCS algorithm.

3 Evaluation

The publications on graph drawing generally include several examples of the presented algorithms. There are several reports on graph drawing systems in the literature. However, it is difficult to do a comparison based on that data, since most systems support only a few algorithm. As of today, there is no common basis. There is a need to measure geometric properties of the layout and collect statistical data from that. There is the need for a generally accepted benchmark. These statistics provide objective criteria.

Nevertheless, there have been several earlier attempts on a comparison of graph drawing algorithms and broader collections of examples. In [36], Tamassia, Di Battista and Batini give a survey of graph layout criteria and algorithms. Fruchterman and Reingold [18] have many examples, and compare their results with drawings from Kamada and Kawai [28]. We have included most of their examples in our database. In [6], Davidson and Harel provide many examples, and compare their algorithm to layouts from force directed algorithms. They did not provide statistics. Most of their examples are included in our database. Harel and Sardas [21] extend the algorithms and compare them with an implementation of PGS. Several algorithms for drawing planar graphs are compared by Jones et al. in [26]. Among them are implementations of PCS and PGS.

Finally, Di Battista et al. [8] provide an experimental comparison of three orthogonal graph drawing algorithms, which have been implemented using *Diagram Server*. *Diagram Server* is a system which is in its objectives comparable to GraphEd. Their approach to layout evaluation is very similar to ours. They measure formal criteria of the drawings and evaluate and rank them according to statistical data collected from their drawings.

3.1 Evaluation with GraphEd

We have tested our algorithms on more than 100 sample graphs [20], both from the literature and our own. The examples consist of unstructured graphs as well as graphs with a special structure (e.g. circles, trees, grids), and with special graph

theoretic properties (directed acyclic graphs, planar graphs). We have restricted ourselves to connected graphs, since most algorithms need connected graphs as input. Moreover, we ran each algorithm on each applicable input graph.

GraphEd contains two special modules for this purpose, *layout suite* and *layout info*. The layout suite runs all applicable layout algorithms on all graphs in a directory. The layout info module collects statistical and geometric data for each specific drawing. In detail, the following data are collected :

- The *number of nodes and edges* in the graph. Their *sum* is taken as a measure for the size of the graph.
- The *area* of the drawing, as well as *width* and *height*. The *density* of the drawing is defined as the area per node.
- The numbers of *bends* and *crossings*.
- The minimum, average, maximum and standard deviation for *edge lengths*, *angles*, *face sizes* and *node distances*. We also recorded the ratio of the maximum and the minimum value.
- The distributions of *edge lengths*, *angles*, *face sizes* and *node distances*.
- The *runtime* of the algorithm.

We have collected data for all of the above algorithms except the PETRI, GG-H and GG-Df algorithms. They were excluded since they need graphs in a special input format, and thus do not fit in our layout suite module.

A separate program puts the data in a readable form. It creates a fact sheet for each applicable algorithm and for each graph that consists of the drawing and the set of statistical data. Further, we create two kinds of diagrams, which we call *short* diagrams (Figure 10) and *long* diagrams (Figure 11).

In short diagrams, pairs of data are visualized that have only one value per graph. Pairs such as *size versus time* or *size versus area* fall in this category. The minimum, average and maximum pairs are placed into one diagram for a better comparison.

Long diagrams are used to visualize those data where there is a list of values for each graph, for example edge lengths or face sizes. Each graph is represented by a curve which shows the data distribution. The values are normalized into a $[0 \ldots 1]$ interval, to hide effects that are generated by the size of the graph or the area of the drawing. To identify global trends, we overlay all curves for a particular algorithm.

The complete set of fact sheets and diagrams is available from [20].

4 Results

From our experiments, we came the following conclusions :

Edge Length Distributions Figure 10 shows the distribution of the edge lengths. Each line in a diagram shows the edge length distribution of a single graph. Its edges are listed on the abscissa, sorted by size. The lengths of the edges are taken relative to the longest edge in the graph, and are plotted on the ordinate.

Force directed drawings have a distribution that is different from all other algorithms. The curves are S-shaped which means that there are only few short edges, many medium sized edges and few long edges. All other algorithms have concave curves with many short edges and few long edges.

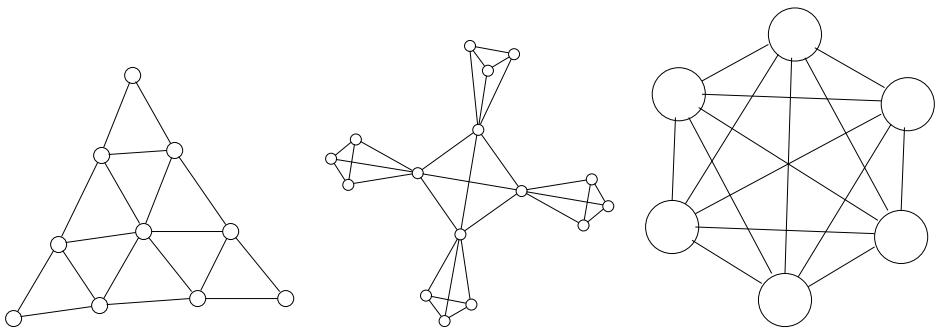

Figure 3: Examples of symmetric graphs, FD-FR algorithm. Note that not all lobes are drawn equally in the graph in the center. The heuristics has run into a local instead of a global optimum.

Also, the curves of the force directed algorithms start at much higher values than the others, which means that the ratio of the longest and the shortest edge is much smaller. In particular, the PGS produces many very short edges, and several long edges. Our experiments indicate that an ideal distribution would start at a high value, has a few short, many medium and a few long edges.

The differences between the two force directed variants is clearly visible in the diagram. The smoother one (FD-K) has a narrower distribution, which is more concentrated towards the upper left corner of the diagram.

Edge Lengths Figure 11 shows the minimum, average and maximum edge lengths of the graphs from our database. In the diagrams, a point on the abscissa corresponds to the size of a graph. Its minimum, average and maximum edge lengths are shown on the ordinate.

In force directed drawings, the ratio of the longest edge and the shortest edge is much smaller than in other algorithms. Their individual differences can also be seen in the diagrams; the minimum, average and maximum edge lengths of FD-K roughly stay at the same level for graphs of all graph sizes. This indicates a very smooth distribution of the nodes, which can also be seen in the examples. In the FD-FR algorithm, they increase with the size of the graph, which is reflected in a less smooth distribution of the nodes in the drawings. The ratio of the longest and the shortest edge roughly stays at the same level for both algorithms.

In all other algorithms, the ratio of the longest and the shortest edge is much higher, and also increases rapidly with the size of the graph. This is mostly consistent with our observation that these algorithms produce a less uniform distribution of nodes and edges.

The *scales* on the ordinates in Figure 11 provide additional information. They show that FD-K produces the shortest edges, followed by FD-FR. All other algorithms produce much longer longest edges. Due to our implementation, the shortest edges for all but the force directed algorithms are of the same length (the standard grid size).

Structure We have found many examples where displaying the structure of a graph is important. For example, force directed drawing algorithms stress symmetric and isomorphic substructures, which others do not reflect (see Figure 3, and Tables 2 and 3).

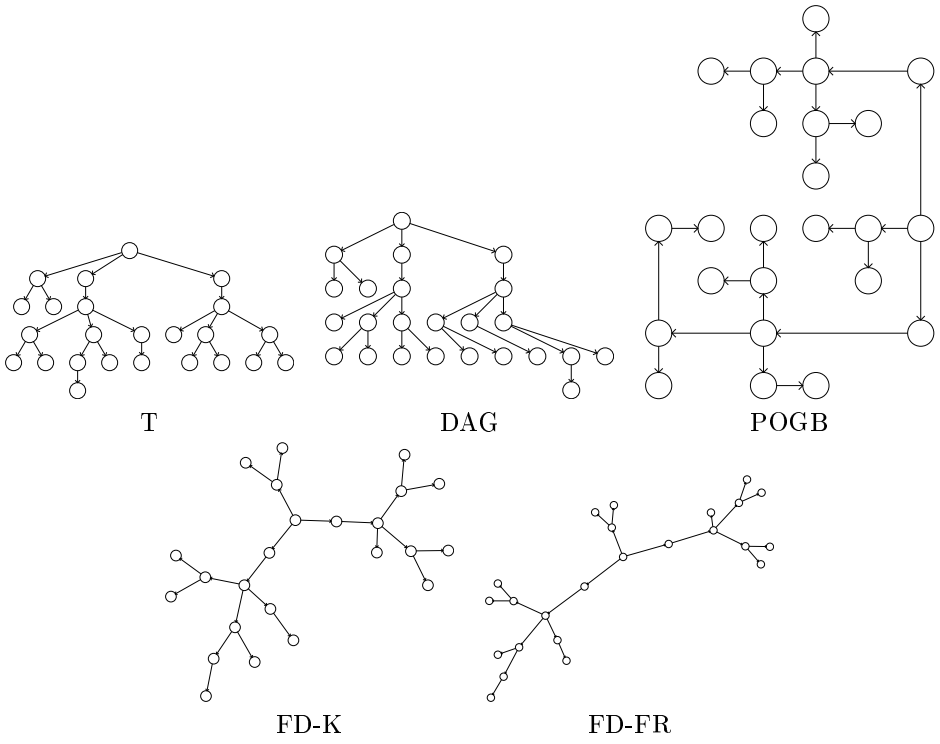

Figure 4: A tree as drawn with various algorithms

The very tree structure is best displayed with the T algorithm (Figure 4). The DAG algorithm partially reflects the graph tree structure, while the POGB algorithm does not recognize the tree structure, and thus does not perform very well. On the other hand, the force directed algorithms perform quite well on trees, since they stress the isomorphic and symmetric substructures, but at the cost of a much higher runtime and distortions in large trees.

The PETRI algorithm draws its nets nicely (Figure 5), whereas all other algorithms fail here.

The graph grammar based algorithms know the hierarchical structure of the graph from the graph grammar. Thus, they are able to reflect this structure in the drawing. Figure 6 shows a drawing of a data flow graph generated by the GG-Df algorithm and compares it to drawings generated by the DAG and FD-FR algorithms. Clearly, only GG-Df displays the structure of the data flow.

Figure 7 shows a drawing of a binary tree drawn by a graph grammar based approach (GG-H), and compares that drawing to T and DAG. Figure 8 shows series parallel graphs as drawn by GG-H, FD-FR, DAG and POGB. Again, the intrinsic structure of the drawing is displayed best by those algorithms which have the best knowledge of the structure.

Orthogonal Edges The POGB algorithm produces the best drawings for planar graphs. Figure 9 shows that this is not only due to the few bends. Additionally, horizontal and vertical edges improve the quality of the drawing a lot.

The POGB algorithm produces drawings with only horizontal and vertical edge segments, and parallel edge segments are always separated by one grid

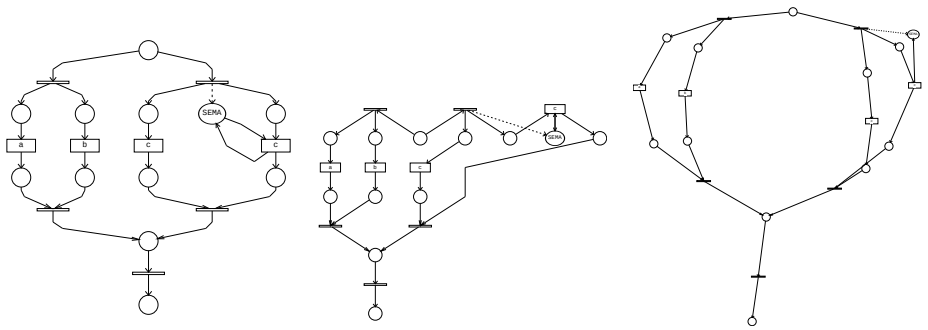

Figure 5: A Petri net, drawn by various algorithms. Only the PETRI algorithm can recognize the Petri net structure in an appropriate way.

unit. Thus, the edges are easier to distinguish as in the PG algorithm, which can produce arbitrary slopes and very narrow parallel edges.

Visibility layouts, where the edges are drawn as straight vertical lines, and the nodes are stretched out horizontally, confirm this experience. It is interesting to see that the POGB algorithm uses a larger area and a longer longest edge. This strongly suggests that these formal criteria do not guarantee a good layout.

Grid A grid can help to produce a clean drawing, as in the POGB algorithm. But it is clearly a source of problems in the PGS algorithm. On the other hand, the force directed algorithms show that a grid is not needed for good results.

Straight Line Edges The force directed algorithms perform well with straight line edges. Obviously, we can easily follow straight line edges with our eyes. On the other hand, straight line edges can be a drawback, as seen in the PCS and PGS algorithms.

We believe that straight line edges can cause trouble if the placement of nodes is further restricted by other criteria. For example, the combinations planarity, grid and straight line edges (PGS) or planarity, convex faces and straight line edges (PCS) are such problematic bundles. This means that further polishing is needed.

Furthermore, the drawings generated by the POGB algorithm show that straight line edges are not always necessary for a nice layout (see Figure 9). A few bends are easily tolerable.

Crossings From the comparison of the DAG approach with the algorithms for planar layouts in Tables 2 and 3, it becomes evident that crossings should be avoided. Also, the force directed algorithms produce less crossings than DAG, but even those few catch the readers eye immediately.

Area Except for the PCS and PGS algorithms, the area consumption of the drawing is usually not critical in our algorithms. Users prefer small area, but other criteria seem to be more important. As already stated above, the POGB algorithm consumes more area than the PG algorithm, but most users prefer the POGB's drawings.

Our experience is that a reasonable distribution of the nodes and the edges usually leads to an acceptable area consumption.

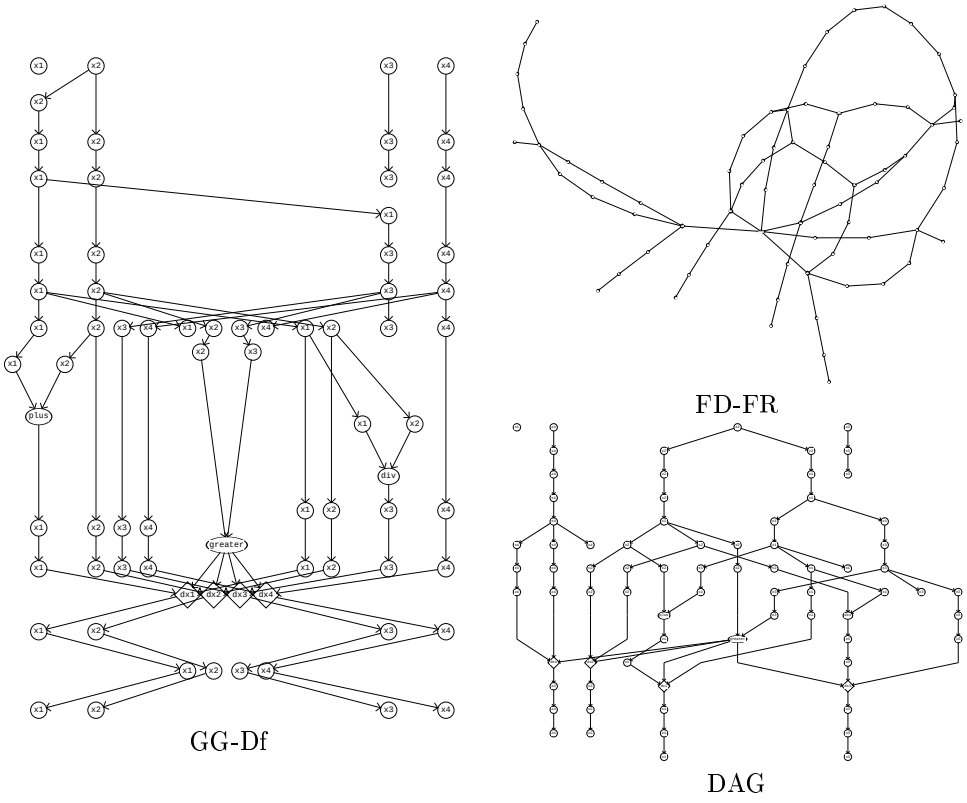

Figure 6: A data flow graph, drawn by various algorithms. Note that only GG-Df preserves the hierarchical structure of the program.

Time The runtime for layout algorithms can be classified into five categories: *very slow*, *slow*, *medium*, *fast* and *very fast*. One should note that these are only approximate descriptions as actual times heavily depend on external issues such as implementation details, operating system or cpu speed.

very slow (GG-H) means that the algorithm usually runs several minutes, or even hours for graphs with size less than 100.

slow (FD-K, FD-FR) means that the algorithm runs from several seconds up to minutes for graphs with size up to 100. The running time grows quickly for larger graphs.

medium (POGB, DAG) means that the algorithm runs from several seconds to several tens of seconds on graphs with size up to 100. The time then increases moderately as the size of the graph grows.

fast (PG) means that the algorithm responds instantaneously for small graphs and takes at most a few seconds for our examples.

very fast (PCS, PGS, PETRI, GG-H) means that the algorithm runs almost instantaneously for our examples.

The graph grammar drawing approaches are very fast if they are used interactively, that is one derivation step at a time. If they need to parse the graph, then they are very slow.

Optimization Generally, we observed that exact optimization is not always necessary. The POGB algorithm draws graphs with a larger area then the PG

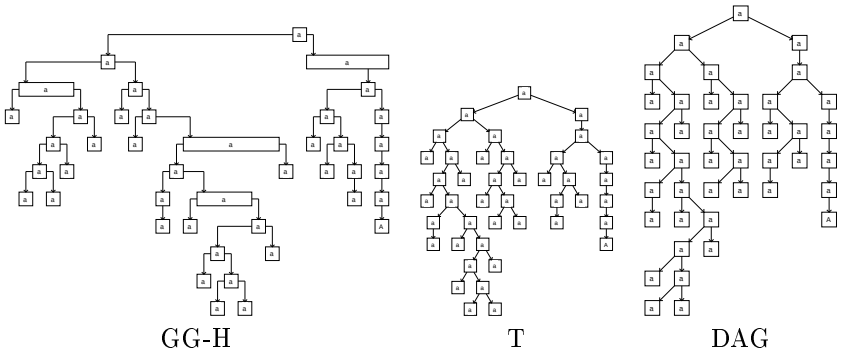

Figure 7: A binary tree drawn by the GG-H, T and DAG algorithms

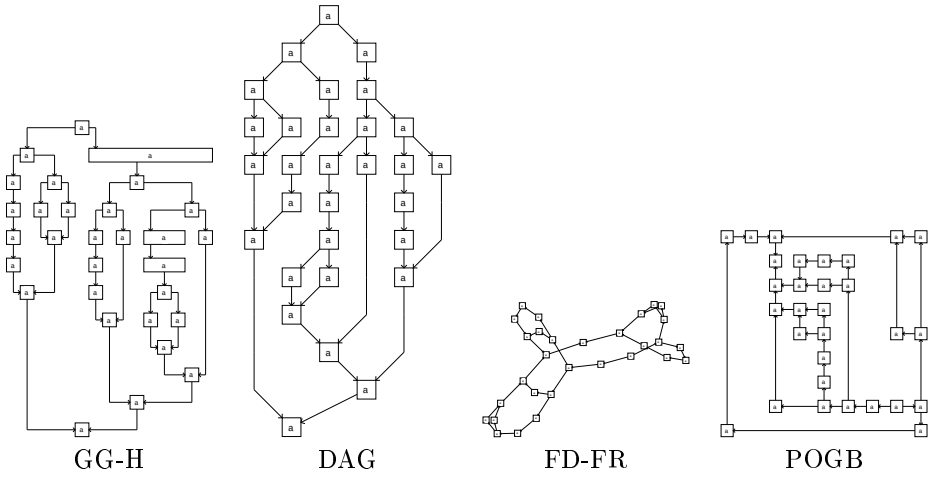

Figure 8: A series parallel graph drawn by the GG-H, DAG, FD-FR and POGB algorithms.

algorithm, and with a larger longest edge. Some criteria, like exact area minimization, would take too long since they are NP-complete.

From our experience, a balanced layout with few extreme values is generally better than a minimized or maximized solution. For example, force directed algorithms draw planar graphs with nearly convex inner faces, but generally much nicer than the PCS algorithm. And their distribution of the edge lengths is more evenly balanced than those of the other criteria.

5 Ranking

From the experiments, our actual ranking of layout criteria is as follows :

1. Distribute the nodes in a uniform fashion.
2. Display the intrinsic structure of the graph.
3. Display symmetric and isomorph substructures of the graph.
4. Use few edge crossings to draw the graph (none if the graph is planar).
5. Use few bends to draw the graph.

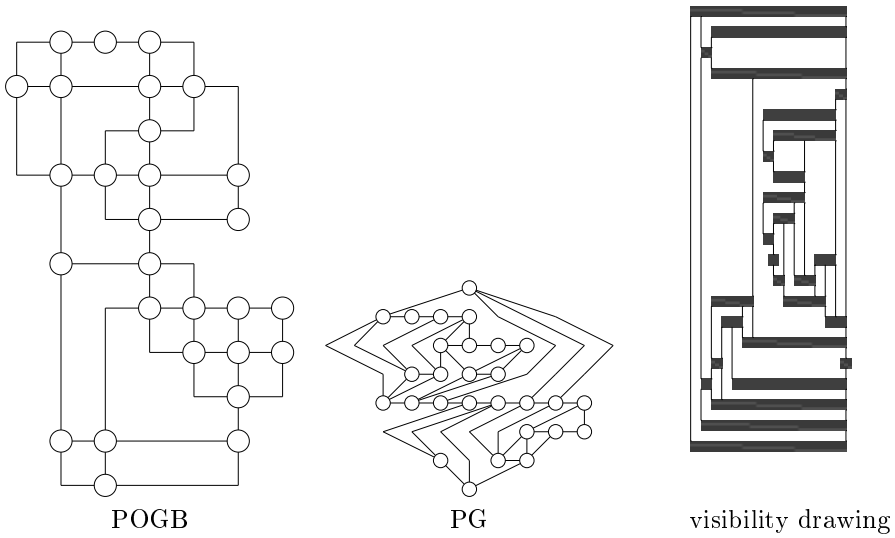

Figure 9: Planar drawings, algorithms POGB (14 bends) and PG (20 bends). At the right, there is a visibility drawing of the same graph; the nodes are stretched to horizontal bars.

6. Place nodes and bends on a grid.

The motivation for the highest rank for the uniform distribution comes from our experiments with the algorithms FD-FR, FD-K and POGB. They profit from a smooth distribution of nodes and edges. The latter one guarantees that parallel edges are separated by at least one grid unit, which gives the drawing a clean appearance. On the other hand, PCS and PGS produce drawings with a very uneven distribution and node clustering.

Displaying the structure is clearly the next point, as one can see in the PETRI and GG-Df algorithms. The success of FD-K and FD-FR is tightly connected with their ability to stress symmetric and isomorphic substructures.

We choose few crossings over few bends, since the PG algorithm usually performs better than the DAG algorithm on the same graph. The latter produces crossings even in planar graphs, which results in a worse layout. Grids are of least importance since they definitivly help algorithms like POGB, DAG or T to produce a clean drawing, but algorithms like FD-FR and FD-K perform well without a grid.

6 Conclusion

Layout algorithms are an ongoing challenge to theoretical computer scientists and designers of user interfaces. With our GraphEd system, we are able to compare different algorithms, and study their advantages and disadvantages. We conclude from our experiments that traditional layout criteria and their ranking must change, and the algorithms must be revised.

Another important issue will be the development of more flexible and application oriented layout algorithms. Most current algorithms (probably with the exception of the *dot* [19] and the *Diagram Server* [9] systems) do not offer much flexibility. They cannot be adapted to the specific needs of the structures that they display. Our data flow and Petri net experiments clearly show the benefit of such approaches.

Techniques such as graph grammars and declarative graph drawing ([12], [5]) are needed to extend the old paradigms. GraphEd's layout algorithms will be refined in

these directions. Other plans include dynamic algorithms, the application of layout algorithms to real world structures, as well as extending our statistics.

Acknowledgements I would like to thank the referees and F.J. Brandenburg, Peter Eades and Brendan Madden for many fruitful comments and discussions, which helped to improve the paper. I would also like to thank all the numerous people who have contributed to build GraphEd and evaluate our layout algorithms.

References

[1] T. Biedl and G. Kant (1994) A better heuristic for orthogonal graph drawings. In *Proceedings of the 2nd Annual European Symposium on Algorithms (ESA '94)*, Lecture Notes on Computer Science **855**, pp. 24–35.

[2] F.J. Brandenburg, M. Himsolt, H.J. Röder & K. Skodinis (1995) *Designing dataflow graphs by graph grammars.* Technical Report **MIP 95-06**, Universität Passau.

[3] N. Chiba, K. Onoguchi & T. Nishizeki (1985) Drawing planar graphs nicely. *Acta Informatica* **22**, pp. 187–201.

[4] M. Chrobak and T.H. Payne (1990) *A linear time algorithm for drawing a planar graph on a grid.* Technical Report **UCR-CS-90-2**, Department of Mathematics and Computer Science, Univ. California, Riverside.

[5] I.F. Cruz and A. Garg (1994) Drawing Graphs by Example Efficiently: Trees and Planar Acyclic Digraphs. In R. Tamassia, I.G. Tollis (editors): *Graph Drawing*, Lecture Notes in Computer Science **894**, pp. 404-415

[6] R. Davidson and D. Harel (1991) *Drawing graphs nicely using simulated annealing.* Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel.

[7] G. Di Battista, P. Eades, R. Tamassia & I.G. Tollis (1994) Algorithms for drawing graphs: An annotated bibliography. *Computational Geometry: Theory and Applications* **4**, pp. 235–282.

[8] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari & F. Vargiu (1995) An experimental comparison of three graph drawing algorithms. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, to appear.

[9] G. Di Battista, G. Liotta & F. Vargiu (1995) Diagram Server. *Journal of Visual Languages and Computation*, this issue.

[10] G. Di Battista, R. Tamassia & I.G. Tollis (1992) Area requirement and symmetry display of planar upward drawings. *Discrete Computational Geometry* **7**, pp. 381–401.

[11] Peter Eades (1984) A heuristic for graph drawing. *Congressus Numerantium* **42**, pp. 149–160.

[12] P. Eades and T. Lin (1994) Integration of Declarative and Algorithmis Approaches for Layout Creation. In R. Tamassia, I.G. Tollis (editors): *Graph Drawing*, Lecture Notes in Computer Science **894**, pp. 376–387

[13] P. Eades and K. Sugiyama (1990) How to draw a directed graph. *Journal of Information Processing* **14**, pp. 424–437.

[14] S. Even (1979) *Graph Algorithms.* Computer Science Press, Potomac, MD.

[15] H. Ehrig, M. Nagl, A. Rosenfeld & G. Rozenberg, editors (1987) *Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science **291**.

[16] H. Ehrig, H.J. Kreowski & G. Rozenberg, editors (1990) *Graph Grammars and Their Application to Computer Science*, Lecture Notes in Computer Science **532**.

[17] H. de Fraysseix, J. Pach & R. Pollack (1990) How to draw a planar graph on a grid. *Combinatorica* **10**, pp. 41–51.

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

[19] E.R. Gansner, E. Koutsofios, S.C. North & K.P. Vo (1993) A technique for drawing directed graphs. *IEEE Transactions on Software Engineering* **19**, pp. 214–230.

[20] GraphEd and related documents are available by anonymous ftp from the site `ftp.uni-passau.de`, `/pub/local/graphed`.

[21] D. Harel and M. Sardas (1995) Randomized graph drawing with heavy-duty preprocessing. *Journal of Visual Languages and Computation*, this issue.

[22] T. Hickl (1995) *Rechtwinkliges Layout von hierarchischen Graphen.* Dissertation, Universität Passau.

[23] M. Himsolt (1993) *Sgraph Programmer's Manual.* Universität Passau.

[24] M. Himsolt (1993) *Konzeption und Implementierung von Grapheneditoren.* Dissertation. Verlag Shaker, Aachen.

[25] M. Himsolt (1994) GraphEd: A Graphical Platform for the Implementation of Graph Algorithms. In R. Tamassia, I.G. Tollis (editors): *Graph Drawing*, Lecture Notes in Computer Science **894**, pp. 182-193.

[26] S. Jones, P. Eades, A. Moran, N. Ward, G. Delott & R. Tamassia (1991) A note on planar graph drawing algorithms. Technical Report **216**, Department of Computer Science, University of Queensland.

[27] M. Jünger and P. Mutzel (1995) Maximum planar subgraphs and nice embeddings: Practial Layout tools. *Algorithmica*, to appear.

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

[29] C. Kosak, J. Marks, S. Shieber (1994) Automating the Layout of Network Diagrams with Specified Visual Organization. *IEEE Transactions on Systems, Man and Cybernetics* **24(3)**, pp. 440–454.

[30] P. Mutzel (1994) The maximum planar subgraph problem. Dissertation, Universität Köln.

[31] E. Reingold and J. Tilford (1981) Tidier drawing of trees. *IEEE Transactions on Software Engineering* **7**, pp. 223–228.

[32] W. Schnyder (1990) Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pp. 138–148.

[33] K. Seisenberger (1991) Termgraph : Ein System zur Zeichnerischen Darstellung von Strukturierten Agenten und Petrinetzen. Diplomarbeit, Universität Passau.

[34] K. Sugiyama, S. Tagawa & M. Toda (1981) Methods for visual understanding of hierarchical systems. *IEEE Transactions on Systems, Man and Cybernetics* **11**, pp. 109–125.

[35] R. Tamassia (1987) On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing* **16**, pp. 421–444.

[36] R. Tamassia, G. Di Battista & C. Batini (1988) Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics* **18**, pp. 61–79.

[37] R. Tamassia, I.G. Tollis (1989) Planar grid embedding in linear time. *IEEE Transactions on Circuits and Systems* **36(9)**, pp. 1230–1234.

[38] J.Q. Walker (1990) A node-positioning algorithm for general trees. *Software – Practice and Experience* **20**, pp. 685–705.

[39] D. Woods (1982) *Drawing Planar Graphs.* PhD thesis, Stanford University.

Algorithm	**Criteria**	**Restrictions**	**Speed**[a]	**Drawing**[b]
FD-K [28]	force directed straight line	connected	slow	$\star\star\star\star$
FD-FR [18]	force directed straight line	connected	slow	$\star\star\star\star$
DAG [34], [13]	hierarchy polyline grid		medium	$\star\star\star$
POGB [35]	planar orthogonal grid polyline bends minimization	planar degree < 4 connected	medium	$\star\star\star\star\star$
PG [39]	planar polyline grid	planar 2-connected	fast	$\star\star\star$
PCS [3]	planar convex faces straight line	planar 2-connected specific	very fast	$\star$ (some $\star\star\star\star$)
PGS [17]	planar straight line grid	planar	very fast	$\star$
T [38]	tree straight line grid	tree	very fast	$\star\star\star\star\star$
PETRI [33]	Petri net	agents	very fast	$\star\star\star\star\star$
GG-H [22]	graph grammars incremental various	derivation	very fast	uncomparable
GG-H [22]	graph grammars parser various	parsed by graph grammar	very slow[c]	uncomparable
GG-Df [2]	graph grammars	input is a derivation	medium[d]	$\star\star\star$

[a] We use a speed rating very slow – slow – medium – fast – very fast (see text).

[b] We use a star rating, where $\star$ is the lowest rating, and $\star\star\star\star\star$ is the highest rating.

[c] The parser is very slow due to its inherently high complexity.
The drawing component itself is fast.

[d] Only prototype implementation available.

Table 1: Layout Algorithms in GraphEd

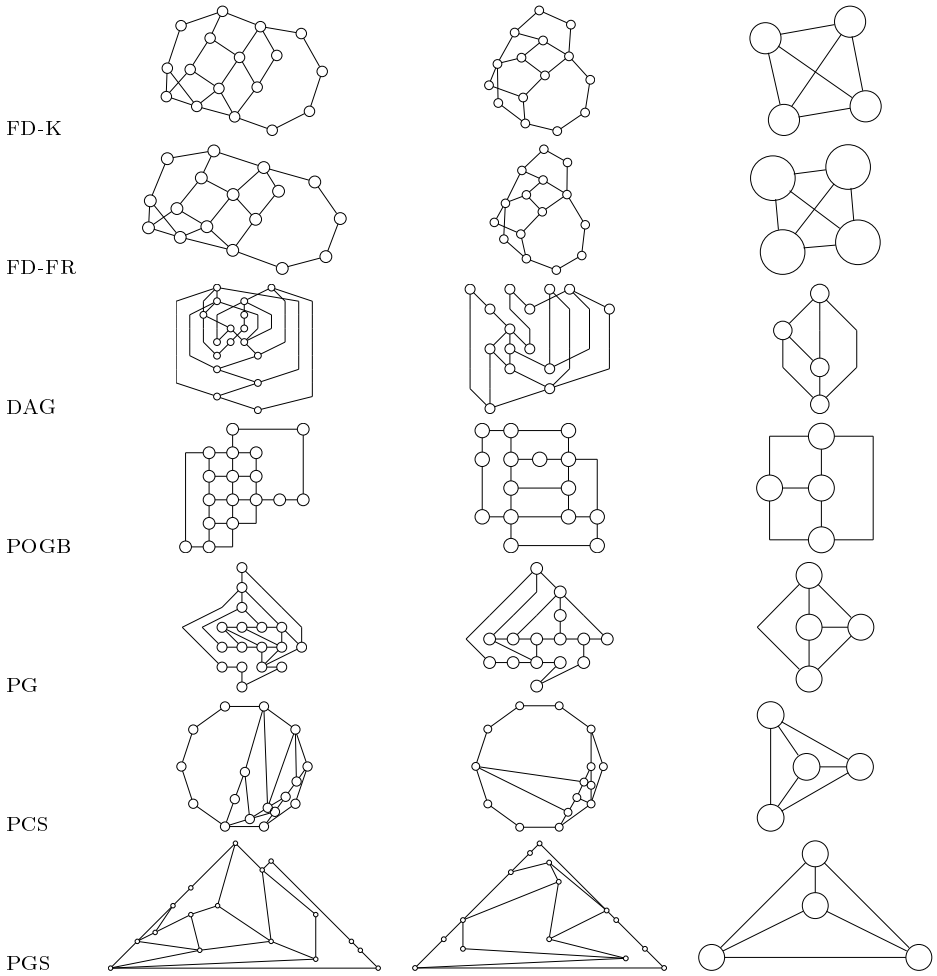

Table 2: Examples of the layout algorithms. Each column shows the same graph.

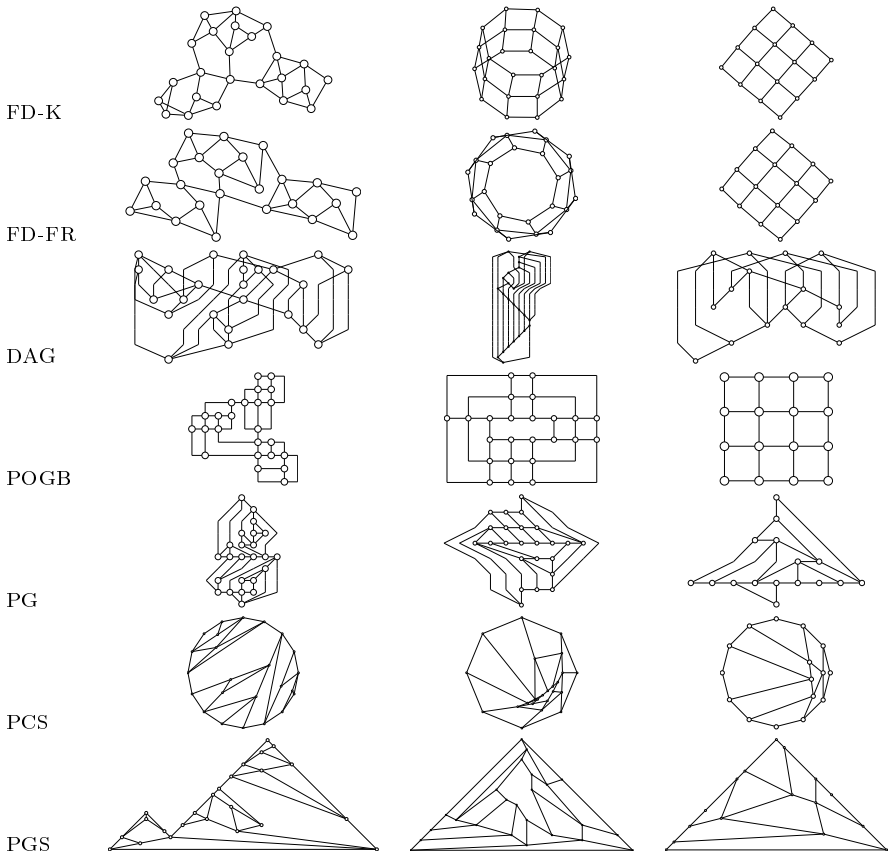

Table 3: More layout examples

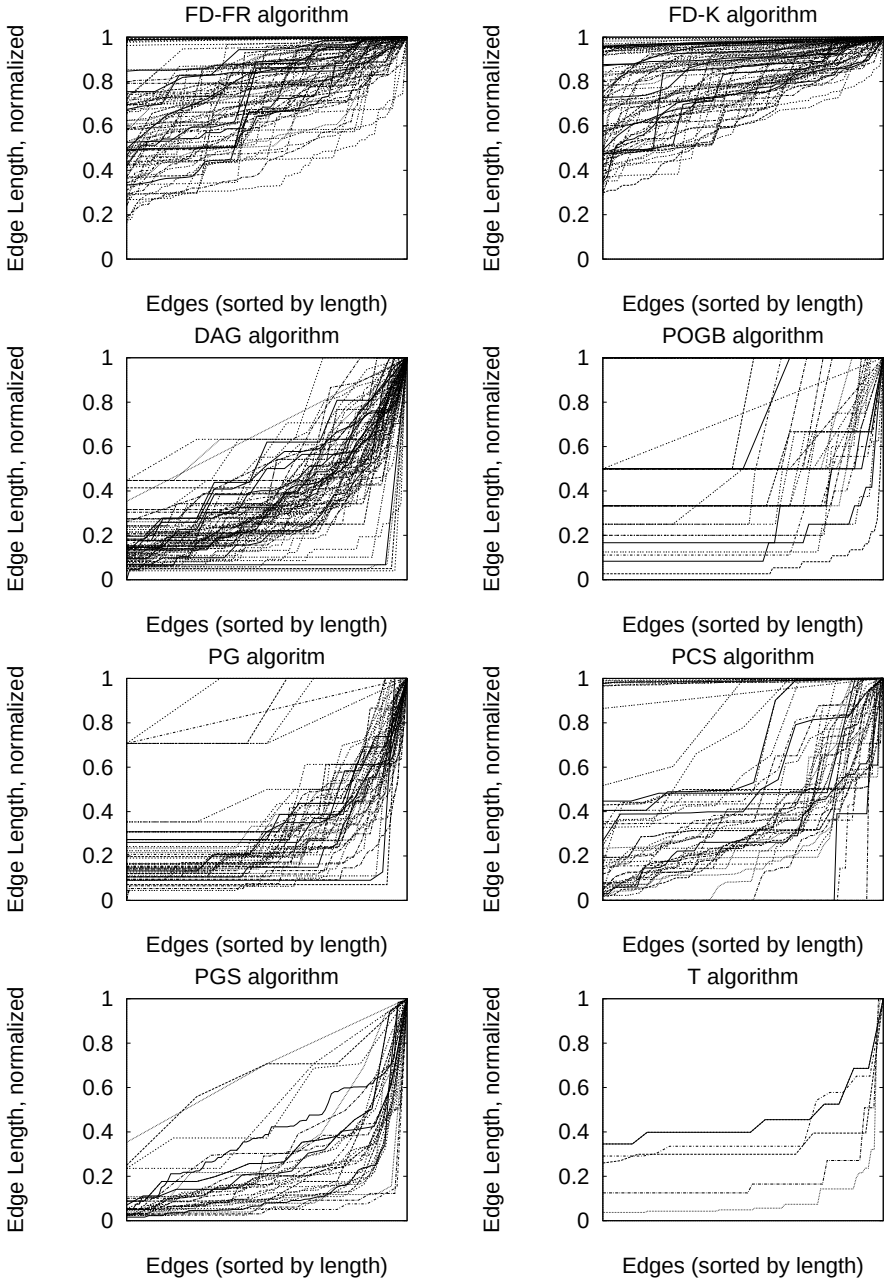

Figure 10: Edge Length Distributions.

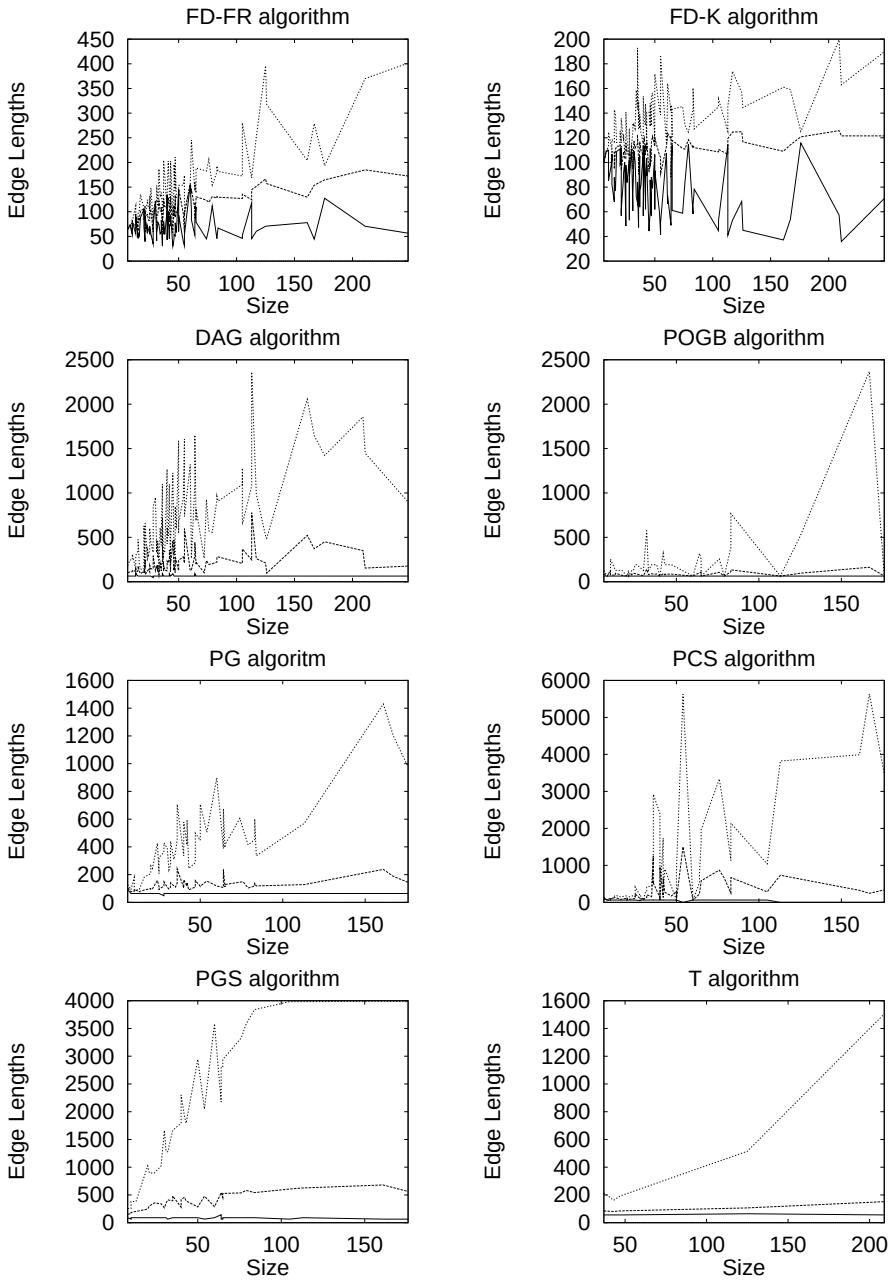

Figure 11: Minimum, average and maximum edge lengths.