Drawing Directed Acyclic Graphs: An Experimental Study

G. Di Battista, A. Garg, G. Liotta, A. Parise,
R. Tamassia, E. Tassinari, F. Vargiu, L. Vismara

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-96-24
October 1996

Drawing Directed Acyclic Graphs: An Experimental Study*

Giuseppe Di Battista
Dipartimento di Discipline Scientifiche
Sezione Informatica
Università degli Studi di Roma Tre
Via della Vasca Navale 84, 00146 Roma, Italy
`dibattista@iasi.rm.cnr.it`

Ashim Garg
Center for Geometric Computing
Department of Computer Science
Brown University
115 Waterman Street, Providence, RI 02912–1910
`ag@cs.brown.edu`

Giuseppe Liotta
Center for Geometric Computing
Department of Computer Science
Brown University
115 Waterman Street, Providence, RI 02912–1910
`gl@cs.brown.edu`

Armando Parise
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
`parise@dis.uniroma1.it`

Roberto Tamassia
Center for Geometric Computing
Department of Computer Science
Brown University
115 Waterman Street, Providence, RI 02912–1910
`rt@cs.brown.edu`

Emanuele Tassinari
Dipartimento Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
`tassinar@dis.uniroma1.it`

Francesco Vargiu
Autorità per l'Informatica
nella Pubblica Amministrazione
Piazzale Kennedy 20, 00144 Roma, Italy
`vargiu@aipa.it`

Luca Vismara
Center for Geometric Computing
Department of Computer Science
Brown University
115 Waterman Street, Providence, RI 02912–1910
`lv@cs.brown.edu`

Abstract

In this paper we consider the class of directed acyclic graphs (DAGs), and present the results of an experimental study on four drawing algorithms specifically developed for DAGs. Our study is conducted on two large test suites of DAGs and yields more than 30 charts comparing the performance of the drawing algorithms with respect to several quality measures, including area, crossings, bends, and aspect ratio. The algorithms exhibit various trade-offs with respect to the quality measures, and none of them clearly outperforms the others.

*Research supported in part by the National Science Foundation under grant CCR–9423847, by the U.S. Army Research Office under grant DAAH04–96–1–0013, by the ESPRIT Long Term Research of the European Community under Project no. 20244 (ALCOM-IT), by the NATO-CNR Advanced Fellowships Programme, and by a gift from Tom Sawyer Software.

1 Introduction

Motivated by applications to information visualization, a large body of graph drawing algorithms has been developed in the last decade. See, e.g., [3, 8, 33].

Many graph drawing papers show sample outputs from prototype implementations and some also provide limited experimental results on small test suites (with fewer than 100 graphs). See, e.g., [7, 16, 18, 24, 25, 26] and the experimental papers in [3, 33]. However, only extensive experimentations can assess the practical performance of graph drawing algorithms in real-life applications. While few studies of this type have been performed in the past, there is now fast growing interest in the important subject of experimental comparative studies of graph drawing algorithms.

1.1 Previous Experimental Work in Graph Drawing

The performance of four planar straight-line drawing algorithms is compared in [22]. The standard deviations in angle size, edge length, and face area are used to compare the quality of the planar straight-line drawings produced. Since the experiments are limited to randomly generated maximal planar graphs, this work gives only partial insight on the performance of the algorithms on general planar graphs.

Himsolt [21] presents a comparative study of twelve graph drawing algorithms. The algorithms selected are based on various approaches (e.g., force-directed, layering, and planarization) and use a variety of graphic standards (e.g., orthogonal, straight-line, polyline). Only three algorithms draw general graphs, while the others are specialized for trees, planar graphs, Petri nets, and graph grammars. The experiments are conducted with the graph drawing system GraphEd [21]. Many examples of drawings constructed by the algorithms are shown, and various objective and subjective evaluations on the aesthetic quality of the drawings produced are given. However, statistics are provided only on the edge length, and few details on the experimental setting are provided. The charts on the edge length have marked oscillations, due to the small size of the test suite (about 100 graphs). This work provides an excellent overview and comparison of the main features of some popular drawing algorithms. However, it does not give detailed statistical results on their performance.

Di Battista et al. [9, 10] present an extensive experimental study comparing four general-purpose graph drawing algorithms. The four algorithms take as input general undirected graphs and construct orthogonal grid drawings. The test graphs are generated from a core set of 112 graphs used in "real-life" software engineering and database applications. The experiments provide a detailed quantitative evaluation of the performance of the four algorithms, and show that they exhibit trade-offs between "aesthetic" properties (e.g., crossings, bends, edge length) and running time. The observed practical behavior of the algorithms is consistent with their theoretical properties.

Brandenburg, Himsolt, and Rohrer [4] compare five "force-directed" algorithms for constructing straight-line drawings of general undirected graphs. The algorithms are tested on a a wide collection of examples and with different settings of the force parameters. The quality measures evaluated are crossings, edge length, vertex distribution, and running time. They also identify trade-offs between the running time and the aesthetic quality of the drawings produced.

Jünger and Mutzel [23] investigate crossing minimization strategies for straight-line drawings of 2-layer graphs, and compare the performance of eight popular heuristics for this problem.

1.2 Our Results

In this paper we consider the important class of directed acyclic graphs (DAGs), and compare the performance of four drawing algorithms specifically developed for them. DAGs are commonly used to model hierarchical structures such as PERT diagrams in project planning, class hierarchies in software engineering, and is-a relationships in knowledge representation systems.

The contributions of this work can be summarized as follows:

- We have developed a general experimental setting for comparing the practical performance of graph drawing algorithms for DAGs. Our setting consists of: (*i*) two large test suites of DAGs, one obtained from the collection of directed graphs submitted to the e-mail graph drawing service at Bell Labs [28], and the other randomly generated by a program that simulates a PERT project planner; (*ii*) a set of quality measures for drawings of DAGs derived from [9].

- Within our experimental setting, we have performed a comparative study of four popular drawing algorithms for DAGs: two of them are based on the layering paradigm [27, 30], while the other two are based on the grid paradigm [12, 14].
- Our comparison highlights how more than ten years of research in this field have produced a complex landscape. Namely, the four algorithms exhibit various trade-offs with respect to the quality measures, and none of them clearly outperforms the others. The sometimes surprising findings of our investigations include:
  - Some algorithms construct very compact drawings at the expense of a relaxed resolution rule that does not consider crossing-crossing and vertex-crossing distances. Other algorithms produce drawings that distribute vertices and crossings with great regularity at the expenses of a larger area requirement.
  - Concerning bends, an algorithm with good theoretical worst-case bounds performs in practice worse than algorithms for which no theoretical bounds are available.
  - Concerning crossings, grid-based algorithms tend to perform worse than layering-based algorithms, where part of the geometry of the drawing is decided at the very first step.
  - The performance of a drawing algorithm on planar DAGs is not a good predictor of the performance of the same algorithm on nonplanar DAGs
  - Algorithms with a topological foundation tend to distribute the bends and the lengths of the edges more evenly.
- Our analysis of the performance of the four algorithms has motivated us to develop a new hybrid strategy for drawing DAGs, which uses a layering-based method to perform the initial planarization and a grid-based method to compute the final drawing. The new strategy performs quite well in practice.
- Any application developer that has to select a drawing algorithm for a given family of DAGs can compare the requirements of the application with our charts and have guidelines to decide which is the best suitable algorithm for his/her purposes.
- We also propose our setting and our experimental results as a workbench to evaluate future algorithms in this field.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In Section 2, we overview the drawing system used for our study. The four algorithms being compared are described in Section 3. Details on the experimental setting are given in Section 4. In Section 5, we summarize our experimental results in 30 charts, and perform a comparative analysis of the performance of the four algorithms. In Section 6, the new hybrid strategy is described and its performance is discussed by means of 6 charts. Open problems are addressed in Section 7. Details on the algorithmic paths used are given in the Appendix.

2 The Graph Drawing Workbench

Our graph drawing tool is *GDW* [5], a system for prototyping and testing graph drawing algorithms. It is the natural evolution of the Diagram Server system [11] toward the realization of an extensible and parametric platform for experimental research on graph drawing.

The user interacts with *GDW* through a multimedia interface. *GDW* presents the algorithms to the user through a *taxonomy* of classes of graphs [2]. The most general class of graphs in the taxonomy is *Multigraph*: a multigraph is a graph that has both directed and non-directed edges. All the other classes in the taxonomy are subclasses of *Multigraph*. Each class is provided with a set of *methods* that map an object of a class into an object of another class. A method is a layout functional step, taken from an existing algorithm. A drawing algorithm A is a sequence of methods that is visually represented on the taxonomy as a path (*algorithmic path*); the edges of the algorithmic path describing A are the methods of A and the vertices are the classes of the taxonomy the methods are associated to. For each method the system can provide one or more bibliographical references.

GDW offers several facilities for randomly generating classes of graphs, such as directed, undirected, and planar graphs. Also, it allows the user to execute an algorithmic path on a given set of graphs and to generate statistics about its behavior with respect to a preselected set of requirements.

3 The Drawing Algorithms Under Evaluation

We have tested four different algorithms for producing drawing of DAGs. These algorithms can be classified into two categories on the basis of their approach to constructing drawings.

Layering-Based: These algorithms construct *layered drawings*, i.e., drawings where the vertices and edge-bends are placed at integer coordinates on a set of horizontal layers, and each edge is drawn as a curve monotonically increasing in the y-direction. Note that in such drawings, even though vertices and edge bends are placed at integer coordinates, the edge crossings can be arbitrarily close to each other or to the vertices and edge bends. These algorithms accept as input directed graphs without any particular restriction (the input directed graph can be planar or not, acyclic or cyclic). For constructing drawings, they generally follow the methodology of Sugiyama et al. [30], which consists of the following three steps:

Step 1 Assign vertices to layers heuristically optimizing some criteria, such as the total edge length.

Step 2 Reduce the crossings among edges by permuting the order of vertices on each layer.

Step 3 Reduce the number of bends by readjusting the position of vertices on each layer.

Because of their generality and conceptual simplicity, these algorithms are very popular among the designers of practical graph drawing systems. Several layering-based algorithms have been designed (see [8] for a detailed bibliography). The above steps have also been investigated separately, and various heuristics have been proposed for each of them [8].

In this paper, we have evaluated and compared the performance of two layering based algorithms: *Dot* and *Layers*.

Dot is a highly optimized algorithm, developed by Koutsofios and North [27] as a successor to *Dag* [17, 18]. *Dot* first constructs a polyline layered drawing of the input directed graph and then, as a final step, converts the polygonal chains representing the edges into smooth curves using splines. An implementation of *Dot* is available at `ftp://ftp.research.att.com/dist/drawdag/`, and this is the implementation we used. However, since all other algorithms considered in this study represent edges as polygonal chains, we decided to analyze the polyline drawing produced by *Dot* and not the final drawing with curved lines.

Layers is the original algorithm by Sugiyama, Tagawa and Toda [30]. For our study we have used the implementation of *Layers* available in *GDW* [5]. The corresponding algorithmic path is given in the Appendix.

Grid-Based: These algorithms accept, as input, a planar st-graph, i.e., a planar DAG with exactly one source and one sink, and construct an *upward grid drawing* of it. In an *upward grid drawing* the vertices, the edge-bends and the edge-crossings are all placed at integer coordinates, and each edge is drawn as a curve monotonically increasing in the y-direction. Although the requirement of having just one source and one sink may appear too restrictive, such directed graphs occur in several practical applications, such as activity planning (where they are called PERT graphs), network flows, etc.

These algorithms are also called *numbering-based* algorithms because they typically construct a numbering of the vertices and faces of the planar st-graph, and compute the coordinates of the vertices

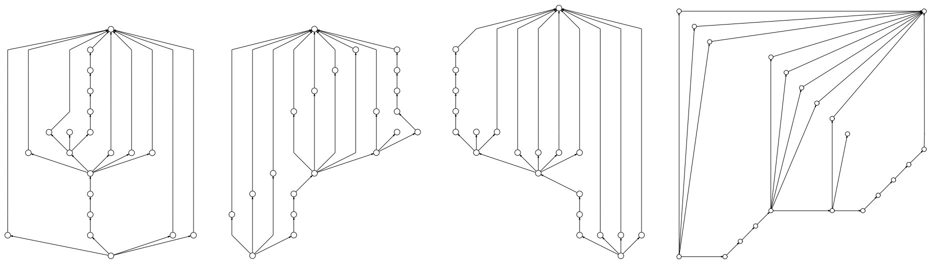

Figure 1: Four drawings of the same North DAG. From left to right, the drawings are produced by *Layers*, *Dot*, *Visibility*, and *Lattice*, respectively. *Dot* converts the polylines to smooth Bezier curves in a postprocessing phase. To facilitate comparison, we have shown here its drawing prior to this smoothening.

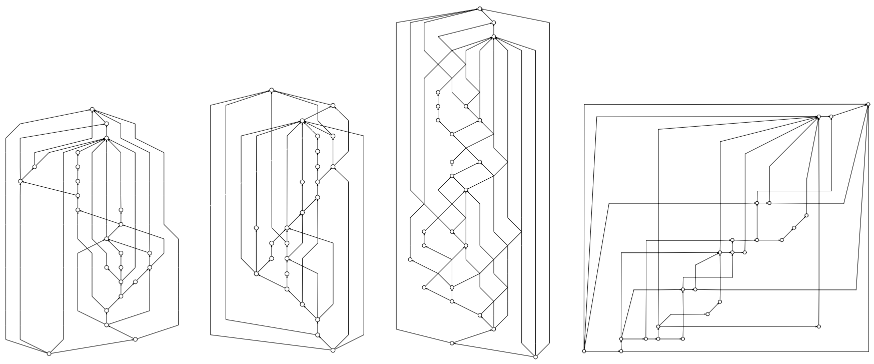

Figure 2: Four drawings of the same Pert DAG. From left to right, the drawings are produced by *Layers*, *Dot*, *Visibility*, and *Lattice*, respectively. *Dot* converts the polylines to smooth Bezier curves in a postprocessing phase. To facilitate comparison, we have shown here its drawing prior to this smoothening.

and bends using this numbering.

The grid-based algorithms have two advantages: first, their performance on planar planar *st*-graphs has been theoretically analyzed, and second, their running times are usually low. The disadvantage is that a nonplanar DAG needs to be converted into a planar *st*-graph, before it can be drawn using these algorithms. This is done by introducing a fictitious vertex for each crossing between two edges. These fictitious vertices are assigned a position on the grid, but are not represented in the final drawing. The simple planarization method we have used for our study is the one described in [12].

The grid-based algorithms that we evaluated and compared fall under two categories:

Visibility Representation-Based: These algorithms use a two-step process for constructing drawings [12, 13]. In the first step, they construct a *visibility representation* of the input planar *st*-graph. (In a visibility representation, vertices and edges are represented as horizontal and vertical line-segments, respectively; two vertices are connected by an edge if and only if they are visible to each other.) In the second step, they construct a polyline drawing of the planar *st*-graph from the visibility representation; this is done by replacing each vertex-segment with a point and by approximating each edge-segment with a polygonal line containing at most two bends.

The visibility representation is constructed using two numberings [15, 31]: a topological numbering of the vertices of the planar *st*-graph, and a topological numbering (in the dual graph) of the faces of the planar *st*-graph. A *topological numbering* of a DAG is such that for every directed edge (u, v), v is assigned a higher number than u.

We have evaluated the performance of an algorithm, called *Visibility*, which follows this approach, and of three variations of it, called *Barycentric Visibility*, *Long Edge Visibility*, and *Median Visibility*. For our study we have used the implementations of these algorithms available in *GDW* [5]. The corresponding algorithmic paths are given in the Appendix. The differences between Algorithms *Visibility*, *Barycentric Visibility*, *Long Edge Visibility*, and *Median Visibility* are in the strategy they use for substituting the vertex and edge-segments of the intermediate visibility representation with the points and polygonal chains of the final drawing; they put the vertex in the middle point of the vertex-segment, in the barycenter of the endpoints of the incident edge-segments, on the endpoint of the longest incident edge-segment, and on the median of the endpoints of the incident edge-segments, respectively.

Poset-Based: These algorithms view planar *st*-graphs as covering graphs of partially ordered sets (*posets*). They exploit the relationship between the upward planarity of DAGs and the order-theoretic properties of planar lattices (see, e.g., [20, 29]).
In our study we have evaluated the performance of one poset-based algorithm: the dominance drawing algorithm of [14]. This algorithm computes two topological numberings of the vertices of the input planar *st*-graph; one numbering gives the x-coordinates of the vertices and bends, and the other gives the y-coordinates. These numberings are obtained by scanning the outgoing edges of each vertex of the planar *st*-graph in the left-to-right and the right-to-left order respectively. For this reason, this algorithm is also known as the *left-right* algorithm. In this paper we have referred to this algorithm as *Lattice*. For our study we have used the implementation of *Lattice* available in *GDW* [5]. The corresponding algorithmic path is given in the Appendix.

Examples of drawings produced by layering-based algorithms and grid-based algorithms are shown in Figs. 1 and 2.

4 Experimental Setting

4.1 Quality Measures Analyzed

The following quality measures of a drawing of a DAG have been considered:

Area: area of the smallest rectangle with horizontal and vertical sides covering the drawing;

Cross: total number of edge-crossings;

TotalBends: total number of edge-bends;

TotalEdgeLen: total edge length;

MaxEdgeBends: maximum number of bends on any edge;

MaxEdgeLen: maximum length of any edge;

UnifBends: standard deviation of the number of edge-bends;

UnifLen: standard deviation of the edge length;

ScreenRatio: deviation from the optimal aspect ratio, computed as the difference between the width/height ratio of the best of the two possible orientations (portrait and landscape) of the drawing and the standard 4/3 ratio of a computer screen.

ResFactor: Inverse of the minimum distance between two vertices, or two edge-crossings, or an edge-crossing and a vertex.

It is widely accepted (see, e.g., [8]) that small values of the above measures are related to the perceived aesthetic appeal and visual effectiveness of the drawing.

The issue of *resolution* of a drawing has been extensively studied, motivated by the finite resolution of physical rendering devices. Several papers have been published about the resolution and the area of drawings of graphs (see, e.g., [1, 6, 14, 19]). The resolution of a drawing is defined as the minimum distance between two vertices. The grid-based algorithms consider edge-bends and edge-crossings as "dummy" vertices for computing the resolution. The layering-based algorithms, however, do not consider the edge-crossings as dummy vertices for computing the resolution. Since the measures Area, TotalEdgeLen and MaxEdgeLen of a drawing depend on its resolution, two drawings can be compared for these measures only if they have the same resolution. ResFactor allows us to scale a drawing D_1 produced by a layering-based algorithm so that it has the same resolution as that of a drawing D_2 produced by a grid-based algorithm; the scaling factor is equal to R_1/R_2, where R_1 and R_2 are the value of ResFactor for D_1 and D_2 respectively.

4.2 Test Suite

The experimental study was performed on two different sets of DAGs, both with a strong connection to "real-life" applications. We considered two typical contexts where DAGs play a fundamental role, namely software engineering and project planning.

The first set of test DAGs are what we call the *North DAGs*. They are obtained from a collection of directed graphs [28], that North collected at AT&T Bell Labs by running for two years *Draw DAG*, an e-mail graph drawing service that accepts directed graphs formatted as e-mail messages and returns messages with the corresponding drawings [27].

Originally, the North DAGs consisted of 5114 directed graphs, whose number of vertices varied in the range $1 \ldots 7602$. However, the density of the directed graphs with a number of vertices that did not fall in the range $10 \ldots 100$ was very low (see also the statistics in [28]); since such directed graphs represent a very sparse statistical population we decided to discard them. Then we noted that many directed graphs were isomorphic; since the vertices of the directed graphs have labels associated with them, the problem is tractable. For each isomorphism class, we kept only one representative directed graph. Also, we deleted the directed graphs where subgraphs were specified as clusters, to be drawn in their own distinct rectangular region of the layout, because constrained algorithms are beyond the scope of this study. This filtering left us with 1277 directed graphs.

Still, 491 directed graphs were not connected and this was a problem for running algorithms implemented in *GDW* (they assume input directed graphs to be connected). Instead of discarding the directed graphs, we followed a more practical approach, by randomly adding a minimum set of directed edges that makes each directed graph connected. Finally, we made the directed graph acyclic, where necessary, by applying some heuristics for inverting the direction of a small subset of edges.

We then ran a first set of experiments and produced the statistics by grouping the DAGs by number of vertices. Although the comparison among the algorithms looked consistent (the produced plots never oddly overlapped), each single plot was not satisfactory, because it showed peaks and valleys. We went back to study the test suite and observed that grouping them by number of vertices was not the best approach. In fact, the North DAGs come from very heterogeneous sources, mainly representing different phases of various software engineering projects; as a result, directed graphs with more or less the same number of vertices may be either very dense or very sparse.

Since most of the analyzed quality measures strongly depend on the number of edges of the DAG (e.g. area, number of bends, and number of crossings), we decided that a better approach was to group the DAGs by number of edges. After some tests, we clustered the DAGs into nine groups, each with at least

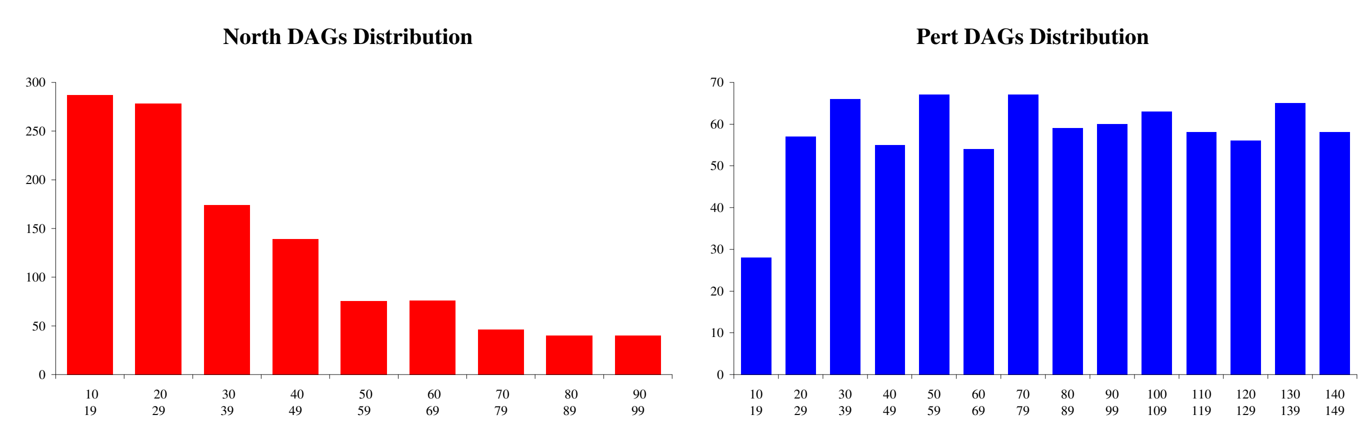

Figure 3: Distributions of DAGs: the x-axis shows the number of edges.

40 DAGs, so that the number of edges in the DAGs belonging to group i, $1 \leq i \leq 9$, is in the range $10i \ldots 10i + 9$ (see Fig. 3). The resulting test suite consists of 1158 DAGs, each with edges in the range $10 \ldots 99$.

The second set of test DAGs are what we call the *Pert DAGs*. Although such DAGs have been randomly generated by one of the facilities of *GDW*, their construction is based on refinement operations typical of project planning.

First, we generated a set of skeleton planar DAGs consisting of a small number of vertices to simulate the initial models of the projects. This was done by randomizing an ear-composition for each DAG. Second, we performed a random sequence of typical planning-refinement steps, i.e., expanding an edge into a series and/or a parallel component and inserting new edges between existing vertices. The inserted edges represent precedences between activities that were not captured by the starting skeleton projects.

The resulting test suite contains 813 DAGs with edges in the range $10 \ldots 150$ and vertices in the range $10 \ldots 100$. As for the North DAGs, we grouped the Pert DAGs by number of edges, so that the number of edges in the DAGs belonging to group i, $1 \leq i \leq 14$, is in the range $10i \ldots 10i + 9$ (see Fig. 3).

The Pert DAGs are generally denser than the North DAGs and they are single-source single-sink. As shown in the next section, there are some quality measures for which the relative performance of the algorithms is different for the North DAGs and the Pert DAGs. Also, the plots obtained for the Pert DAGs are in general smoother, reflecting the relative uniformity of the statistical population.

5 Analysis of the Experimental Results

Algorithms *Dot*, *Layers*, *Visibility* (with its variations *Barycentric Visibility*, *Long Edge Visibility*, *Median Visibility*), and *Lattice* were executed on every North DAG and every Pert DAG, and the data for all ten quality measures were collected. Because of the different nature of the two test suites, we compared the performance of the algorithms for the North DAGs and the Pert DAGs separately. In addition, since the quality measures Area, TotalEdgeLen, and MaxEdgeLen depend upon the resolution of the drawings, we compared the layering-based and grid based algorithms separately for these three quality measures. The other seven quality measures do not depend upon the resolution, so we compared all four algorithms together for them. This gave us a total of 30 comparison charts. Figures 4– 7 display the comparison charts showing the average values for the quality measures; the left column of these figures contains the charts for the North DAGs, and the right column contains the ones for the Pert DAGs. The x-axis of each chart shows the number of edges. The average is computed over each group of DAGs with number of edges in the range $10 \ldots 19$, $20 \ldots 29$, etc.

We started the experimental analysis by comparing the behavior of *Visibility* and its variations *Barycentric Visibility*, *Long Edge Visibility*, and *Median Visibility*. As a result, we found that the behavior of the visibility representation-based algorithms is almost identical for all the quality measures, with Algorithm *Visibility* performing slightly better than the others. An example of this phenomenon is given in the bottom charts of Fig. 5, which show the behavior of the four visibility representation-based algorithms for the measure TotalBends. In order to improve the readability of the other charts and to simplify the presentation of the experimental results, we have used Algorithm *Visibility* as the representative visibility representation-based algorithm.

The analysis of the performance of the four algorithms for each quality measure, and for each set of input DAGs is summarized below:

Area: (see Fig. 4) *Dot* performs better than *Layers*, and *Lattice* performs better than *Visibility* for both the North DAGs and the Pert DAGs. While for the North DAGs, the plots grow linearly for #edges in the range $10 \ldots 60$, for the Pert DAGs they show quadratic growth in the entire range. Also observe that the difference in the performance of the two grid-based algorithms is significant for DAGs with more than 75 edges, whereas the two layering-based algorithms perform about the same in the entire range.

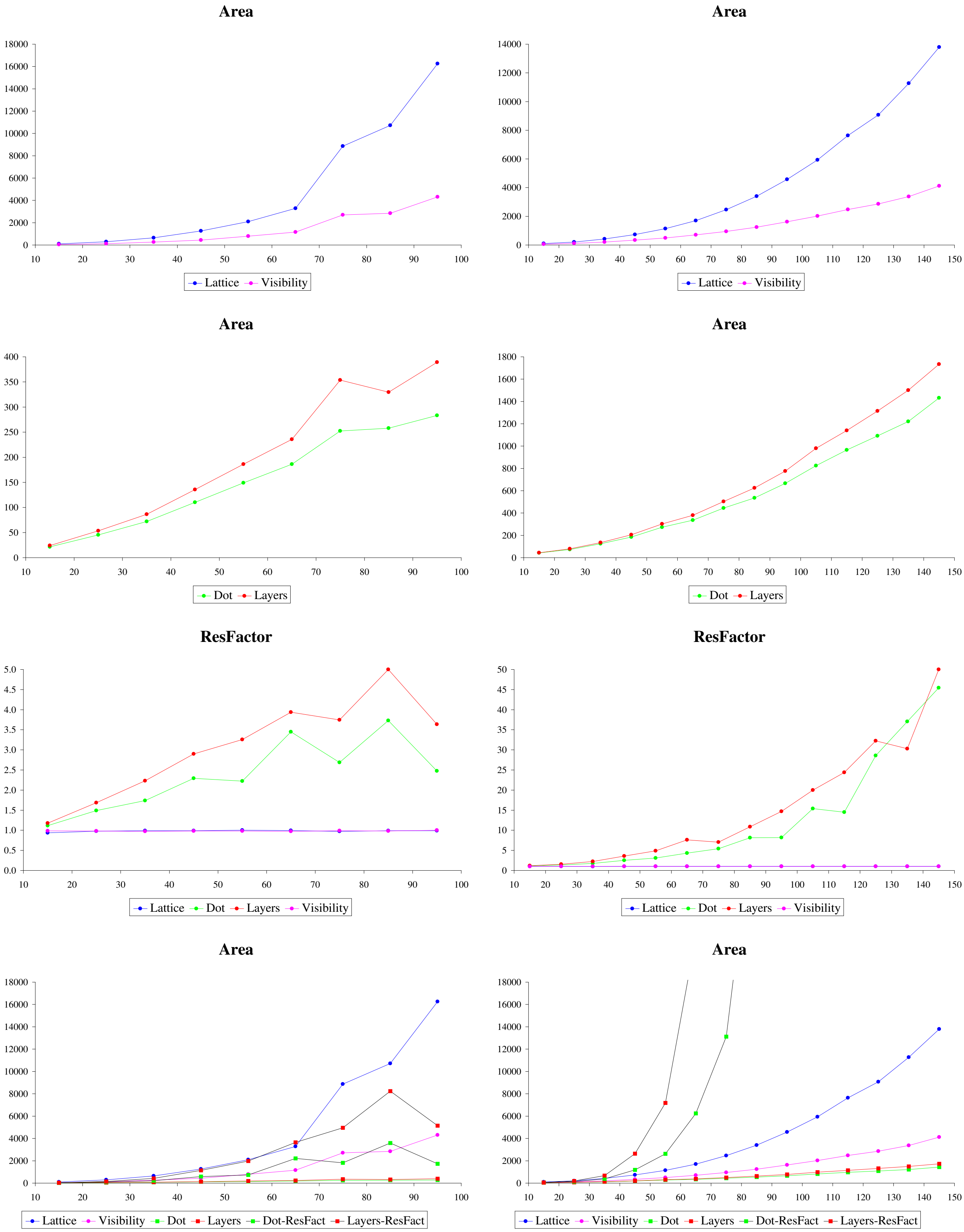

Figure 4: Comparison charts: the x-axis shows the number of edges.

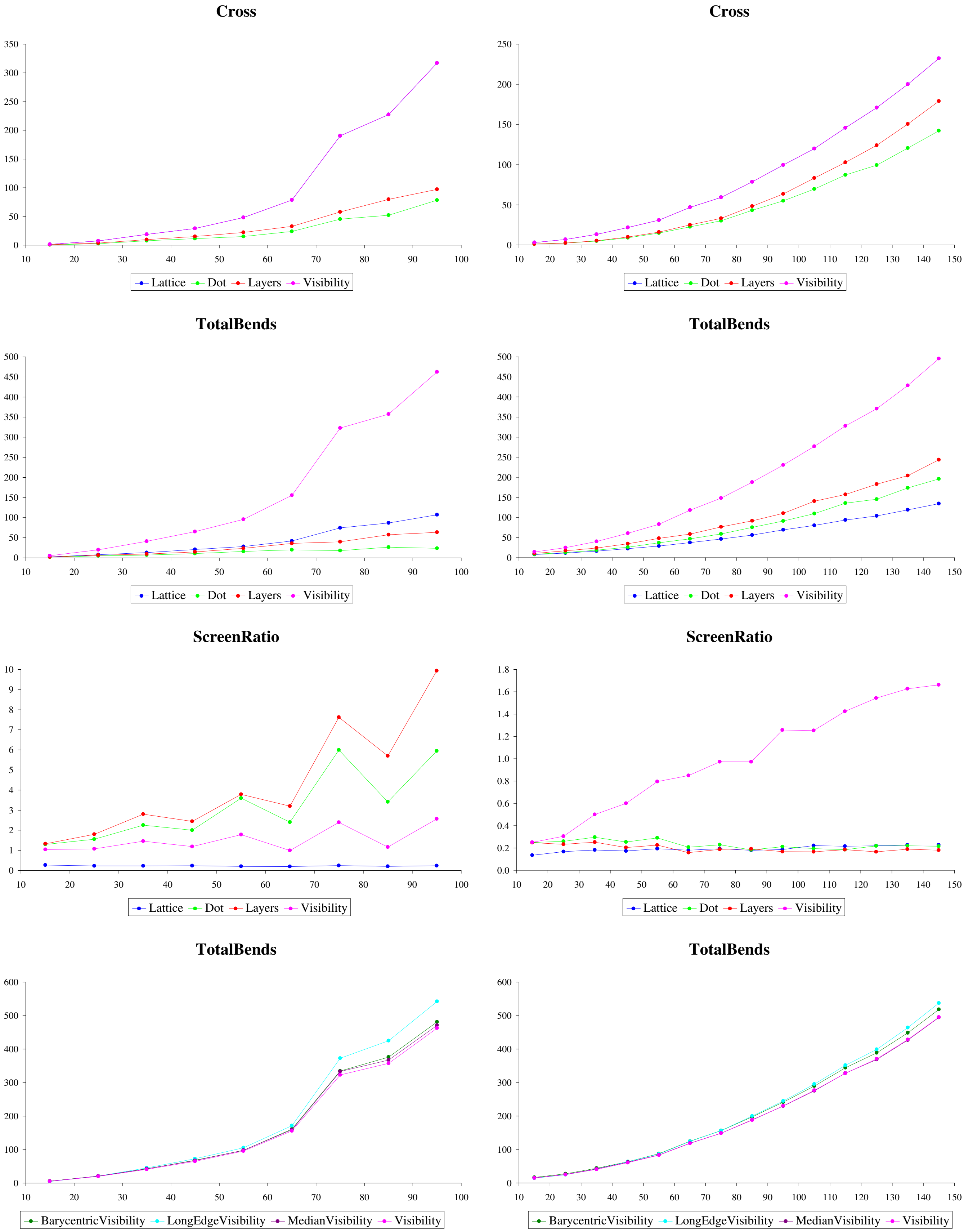

Figure 5: Comparison charts: the x-axis shows the number of edges.

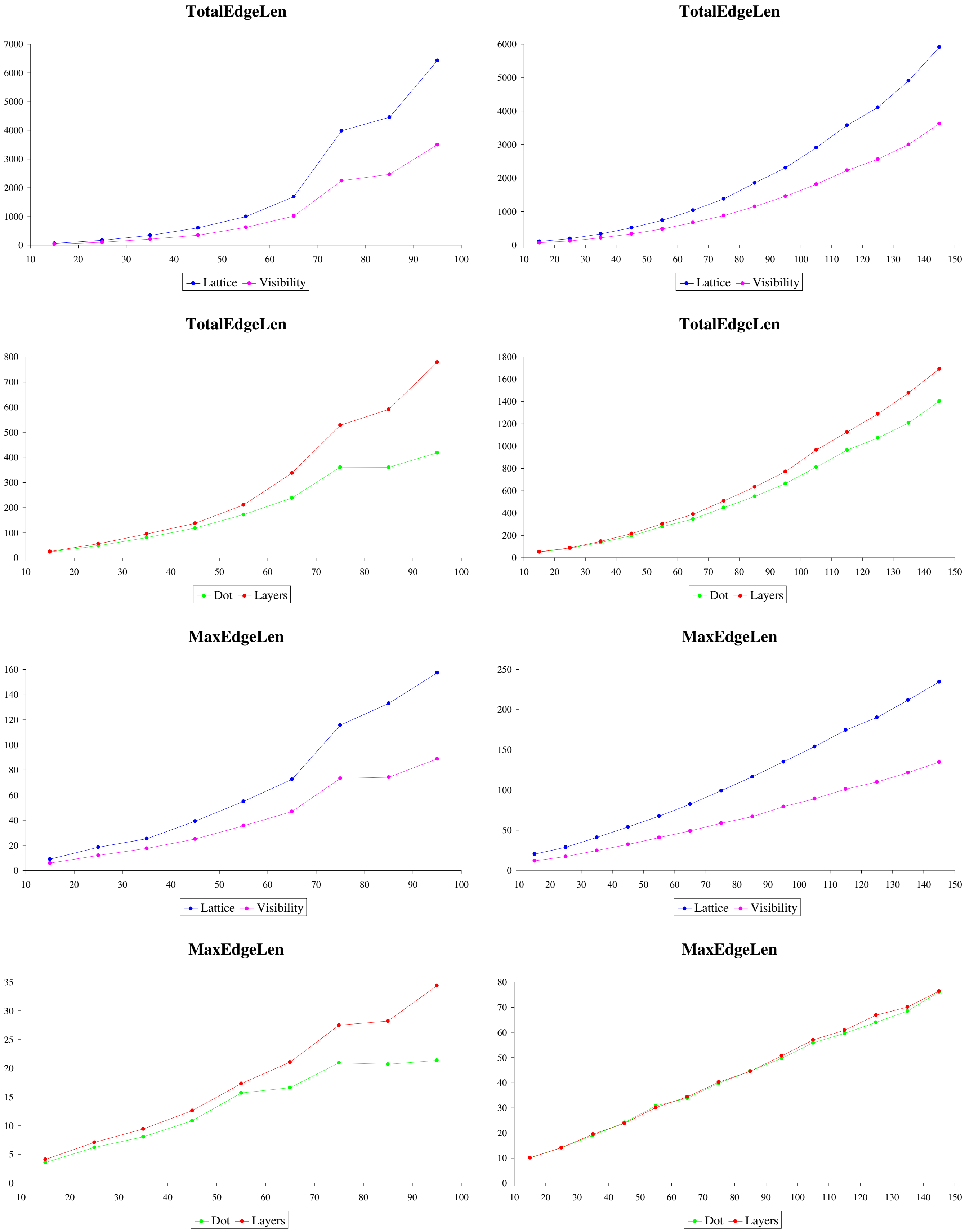

Figure 6: Comparison charts: the x-axis shows the number of edges.

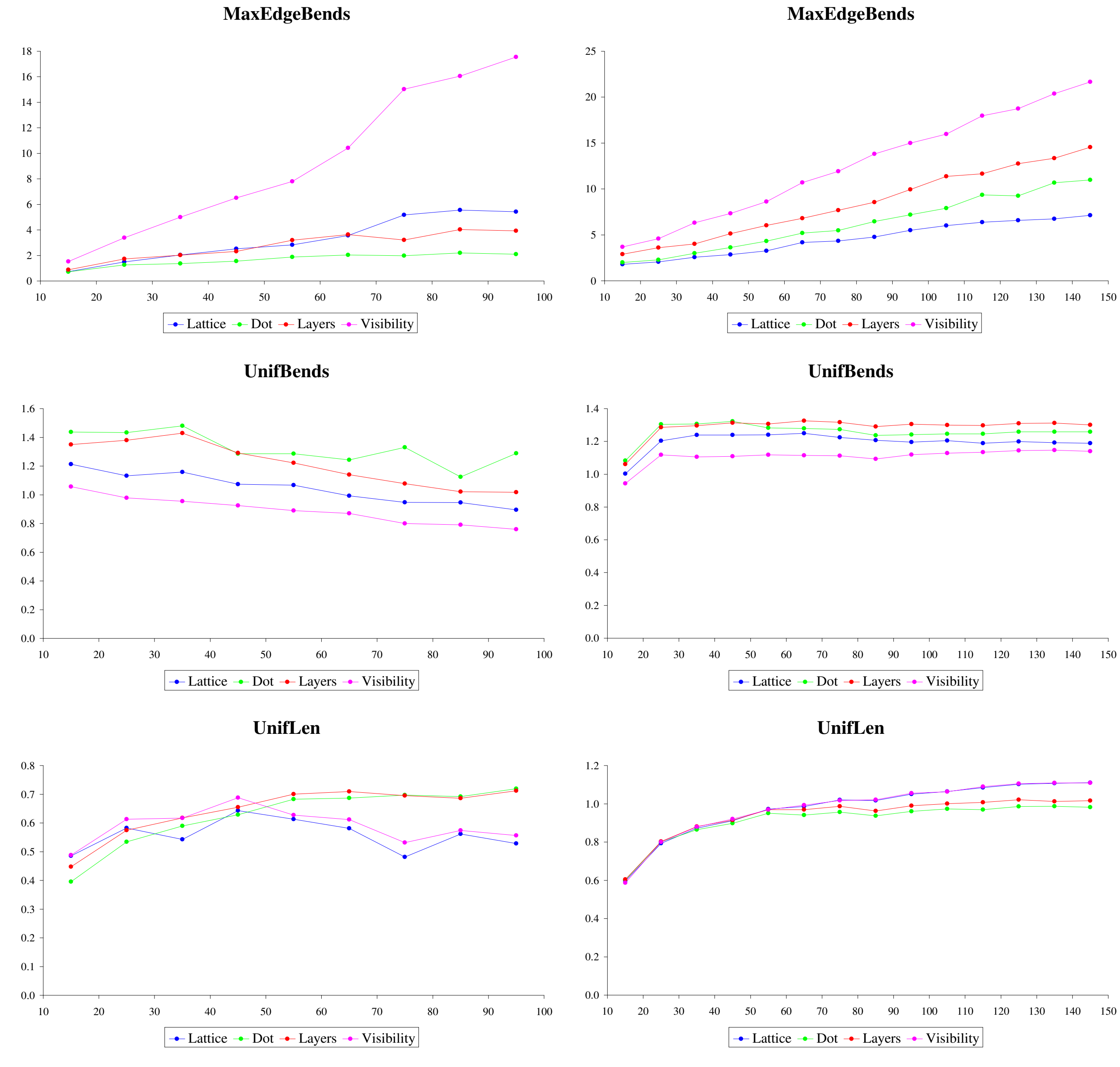

Figure 7: Comparison charts: the x-axis shows the number of edges.

ResFactor: (see Fig. 4) Not surprisingly, ResFactor is equal to one for the grid-based algorithms in the entire range. On the other hand, the layering-based algorithms tend to have a non-constant ResFactor. This reflects the fact that they do not take edge-crossings into consideration for defining the resolution. The bottom charts of Fig. 4 show a comparative study of the area of the drawings produced by the four algorithms. Note that there are two plots for each layering-based algorithm. They show two different measures for the area: one takes ResFactor into account, and the other does not. Note that the plots that take ResFactor into account are comparable with the plots of the grid-based algorithms.

Cross: (see Fig. 5) Since *Lattice* and *Visibility* use the same planarizer (Method `MakeSTPlanar` of *GDW*), the drawings produced by them have the same number of edge-crossings. All the algorithms have quadratic behavior for both sets of DAGs. *Dot* has the best performance among the four. The difference between the performance of the layering-based and the grid-based algorithms reduces considerably for the Pert DAGs. Also observe that the slope of the plots is steeper for the Pert DAGs. This reflects the fact that the Pert DAGs are in general denser than the North DAGs and that the number of edge-crossings tend to increase with the ratio #edges/#vertices.

TotalBends: (see Fig. 5) The performance of *Visibility* is unsatisfactory. For the North DAGs, the plots of the other three algorithms grow almost linearly for #edges up to 65. After that, *Dot* is clearly the best. The experimentation with the Pert DAGs produced a surprising result. Namely, *Lattice* outperforms the layering-based algorithms while *Visibility* has still the worst behavior. As for measure Cross, the slope of the plots is steeper for the Pert DAGs. Note, however, that the behavior of *Lattice* seems to be quite independent from the density of the input DAG, at least for DAGs with up to 75 edges.

ScreenRatio: (see Fig. 5) *Lattice* seems to be the algorithm of choice with respect to this quality measure. All the algorithms have a better performance for the Pert DAGs. We believe that this a consequence of the relative density of the Pert DAGs; the drawings tend to spread in both the x- and y-dimension.

TotalEdgeLen and MaxEdgeLen: (see Fig. 6) These two measures are dependent on ResFactor. Therefore, we compared the performance of the layering-based and grid-based algorithms separately. *Dot* performs better than *Layers*, and *Visibility* performs better than *Lattice* for both the North DAGs and the Pert DAGs.

MaxEdgeBends: (see Fig. 7) Quite interestingly, the plots grow linearly for the Pert DAGs for all four algorithms. While *Dot* has the best performance for the North DAGs, *Lattice* is the best for the Pert DAGs. The overall performance of the algorithms is much better for the North DAGs than for the Pert DAGs. Again, we believe that this is because the North DAGs are in general sparser.

6 Cross-Fertilization of Grid- and Layering-Based Algorithms

The analysis of the experimental results of *Dot*, *Layers*, *Visibility*, and *Lattice* clearly shows that the layering-based algorithms (*Dot* and *Layers*) produce drawings with fewer crossings than the grid-based algorithms (*Visibility* and *Lattice*). This indicates that the crossing reduction step of the layering-based algorithms is more effective than the simple planarization strategy [12] used in *Visibility* and *Lattice*. On the other hand, *Visibility* and *Lattice* perform well with respect to other quality measures (see Section 5).

The above considerations suggest the development of a hybrid strategy that substitutes the original planarization step of *Visibility* and *Lattice* with the crossing reduction step of *Layers* (we choose *Layers* over *Dot* for simplicity of implementation). More specifically, we first execute the crossing reduction step of *Layers* and then visit the resulting drawing, replacing each crossing with a fictitious vertex. This planarizes the input graph. Finally, we execute the remaining algorithmic steps of *Visibility* and *Lattice*. The new drawing algorithms so obtained will be called *VisibilityLayers* and *LatticeLayers*, respectively.

The charts in Fig. 8 compare algorithms *Dot*, *Layers*, *Visibility*, *Lattice*, *VisibilityLayers*, and *LatticeLayers* with respect to quality measures Area, TotalBends, and MaxEdgeBends: the left column of the figure contains the charts for the North DAGs, while the right column contains the charts for the Pert DAGs.

Algorithms *VisibilityLayers* and *LatticeLayers* always perform better than their "parent algorithms" *Visibility* and *Lattice*, respectively. In particular, we observe the following:

Area: (see Fig. 8) The improvement of *VisibilityLayers* and *LatticeLayers* over *Visibility* and *Lattice* is especially significant for the North DAGs, where it ranges between 30% and 50%.

TotalBends: (see Fig. 8) Again, the improvement is especially significant (about 50%) for the North DAGs. Also, while *Layers* is always better than *Lattice*, we have that *LatticeLayers* is slightly better than *Layers* for the North DAGs with more than 70 edges.

MaxEdgeBends: (see Fig. 8) Analogous considerations to those for TotalBends apply. Also, the improvement of *VisibilityLayers* over *Visibility* is substantial.

The analysis of the charts for the other quality measures shows similar trends.

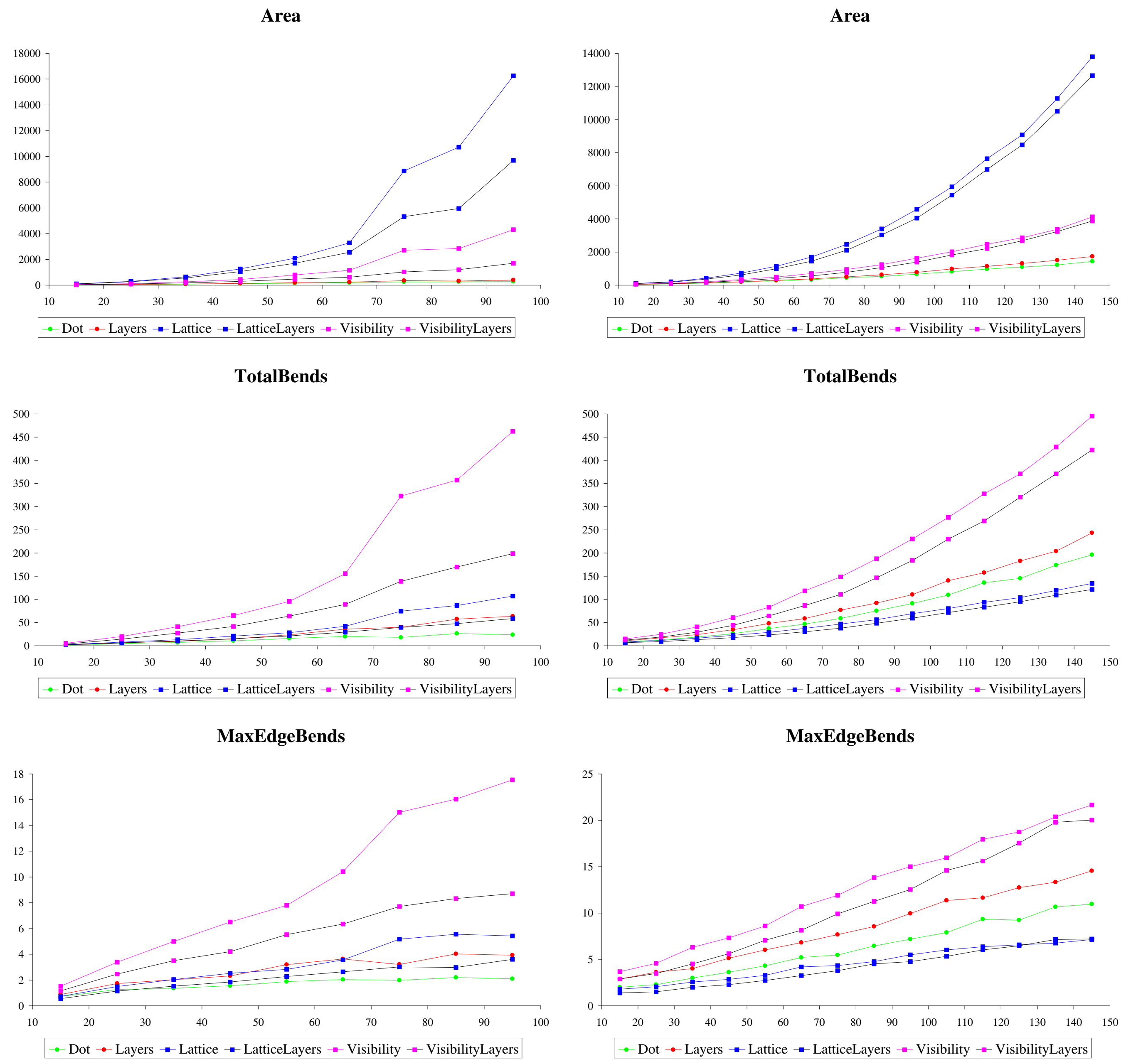

Figure 8: Comparison charts: the x-axis shows the number of edges.

We conclude this section by observing that the performance of the grid-based algorithms (*Visibility* and *Lattice*) is strongly influenced by the number of crossings introduced in the planarization step.

7 Open Problems

Our experiments lead to many interesting theoretical and practical questions:

- The angular resolution of a drawing is the magnitude of the smallest angle between any two edges incident on a vertex. The readability of a drawing can be improved by increasing its angular resolution. Unfortunately, not much is known either theoretically or empirically about the angular resolution of drawings of directed graphs. This issue is worth exploring.

- *Dot* in a final step converts the polylines into Bezier curves using splines. This has a dramatic impact on the quality of the drawing. Similarly, we believe that the performance of several algorithms, such as *Visibility*, can be improved by a postprocessing "beautification" step. For example, it would be interesting to study bend-stretching techniques [32] that reduce the bends by doing local transformations.
- Similarly, the role of the preprocessing step should also be studied. In particular, the performance of grid-based algorithms can be improved by using a more sophisticated planarizer.

Acknowledgements

We are grateful to Stephen North for many discussions about *Dot* and the collection of directed graphs. We thank Petra Mutzel for answering our questions about the statistics on such graphs. Finally we thank Paola Vocca for her encouragement during the writing of this paper.

References

[1] P. Bertolazzi, R. F. Cohen, G. Di Battista, R. Tamassia, and I. G. Tollis. How to draw a series-parallel digraph. *Internat. J. Comput. Geom. Appl.*, 4:385–402, 1994.

[2] P. Bertolazzi, G. Di Battista, and G. Liotta. Parametric graph drawing. *IEEE Trans. Softw. Eng.*, 21(8):662–673, 1995.

[3] F. J. Brandenburg, editor. *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[4] F. J. Brandenburg, M. Himsolt, and C. Rohrer. An experimental comparison of force-directed and randomized graph drawing algorithms. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 76–87. Springer-Verlag, 1996.

[5] L. Buti, G. Di Battista, G. Liotta, E. Tassinari, F. Vargiu, and L. Vismara. GD-Workbench: A system for prototyping and testing graph drawing algorithms. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *LNCS*, pages 111–122. Springer-Verlag, 1996.

[6] M. Chrobak, M. T. Goodrich, and R. Tamassia. Convex drawings of graphs in two and three dimensions. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages 319–328, 1996.

[7] R. Davidson and D. Harel. Drawing graphs nicely using simulated annealing. Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 1989.

[8] 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.

[9] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of three graph drawing algorithms. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 306–315, 1995.

[10] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 1996 (to appear). `http://www.cs.brown.edu/cgc/papers/dglttv-ecfgd-96.ps.gz`.

[11] G. Di Battista, G. Liotta, and F. Vargiu. Diagram Server. *J. Visual Lang. Comput.*, 6(3):275–298, 1995. (Special Issue on Graph Visualization, I. F. Cruz and P. Eades, editors).

[12] G. Di Battista, E. Pietrosanti, R. Tamassia, and I. G. Tollis. Automatic layout of PERT diagrams with XPERT. In *Proc. IEEE Workshop on Visual Languages (VL '89)*, pages 171–176, 1989.

[13] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoret. Comput. Sci.*, 61:175–198, 1988.

[14] G. Di Battista, R. Tamassia, and I. G. Tollis. Area requirement and symmetry display of planar upward drawings. *Discrete Comput. Geom.*, 7:381–401, 1992.

[15] G. Di Battista, R. Tamassia, and I. G. Tollis. Constrained visibility representations of graphs. *Inform. Process. Lett.*, 41:1–7, 1992.

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

[17] 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.

[18] 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.

[19] A. Garg and R. Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In J. van Leeuwen, editor, *Algorithms (Proc. ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 12–23. Springer-Verlag, 1994.

[20] A. Garg and R. Tamassia. Upward planarity testing. *Order*, 12:109–133, 1995.

[21] M. Himsolt. Comparing and evaluating layout algorithms within GraphEd. *J. Visual Lang. Comput.*, 6(3), 1995. (Special Issue on Graph Visualization, I. F. Cruz and P. Eades, editors).

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

[23] M. Jünger and P. Mutzel. Exact and heuristic algorithms for 2-layer straightline crossing minimization. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes in Computer Science*, pages 337–348. Springer-Verlag, 1996.

[24] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996. (Special Issue on Graph Drawing, G. Di Battista and R. Tamassia, editors).

[25] T. Kamada. *Visualizing Abstract Objects and Relations.* World Scientific Series in Computer Science, 1989.

[26] G. Kant. *Algorithms for Drawing Planar Graphs.* PhD thesis, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[27] E. Koutsofios and S. North. Drawing graphs with *dot*, 1993. *dot* user's manual. `ftp://ftp.research.att.com/dist/drawdag/`.

[28] S. North. 5114 directed graphs, 1995. Manuscript. `ftp://ftp.research.att.com/dist/drawdag/`.

[29] I. Rival. Reading, drawing, and order. In I. G. Rosenberg and G. Sabidussi, editors, *Algebras and Orders*, pages 359–404. Kluwer Academic Publishers, 1993.

[30] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.

[31] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.*, 1(4):321–341, 1986.

[32] R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.

[33] R. Tamassia and I. G. Tollis, editors. *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

A The Algorithmic Paths

In this appendix we give the algorithmic paths in *GDW* that we used for our study.

A.1 Algorithmic Path `Layers`

```
Layers = multigraph > IsDirected > digraph > IsAcyclicDigraph > acyclicdigraph >
MakeKLayered > klayered > MakeProperKLayered > properklayered > Layers > straightLine >
DoPolygonal > polygonal
```

In the algorithmic paths language of *GDW*, `IsDirected` is a step that tests if a multigraph is directed and, in case, maps it into a digraph. Also, `MakeKlayered` maps an acyclic digraph to a *k-layered* graph by assigning a layer to each vertex. A *proper k-layered* graph is k-layered and such that each edge spans exactly two layers.

A.2 Visibility Representation-Based Algorithmic Paths

```
Visibility = multigraph > IsDirected > digraph > IsAcyclicDigraph > acyclicdigraph >
MakeSTDigraph > stdigraph > MakeSTPlanar > planarstdigraph > MakeDirVisibilityRepr >
directedvisibilityrepresentation > MakePolygonalPmed > polygonal
```

```
Barycentric Visibility = multigraph > IsDirected > digraph > IsAcyclicDigraph >
acyclicdigraph > MakeSTDigraph > stdigraph > MakeSTPlanar > planarstdigraph >
MakeDirVisibilityRepr > directedvisibilityrepresentation > MakePolygonalBaryc >
polygonal
```

```
Long Edge Visibility = multigraph > IsDirected > digraph > IsAcyclicDigraph >
acyclicdigraph > MakeSTDigraph > stdigraph > MakeSTPlanar > planarstdigraph >
MakeDirVisibilityRepr > directedvisibilityrepresentation > MakePolygonalLongedge >
polygonal
```

```
Median Visibility = multigraph > IsDirected > digraph > IsAcyclicDigraph >
acyclicdigraph > MakeSTDigraph > stdigraph > MakeSTPlanar > planarstdigraph >
MakeDirVisibilityRepr > directedvisibilityrepresentation > MakePolygonalMedian >
polygonal
```

The `MakeSTPlanar` step is needed to convert the input graph to a planar *st*-graph. Observe that the four paths differ in the `MakePolygonal` step. Namely, in the four algorithms, `MakePolygonalPmed`, `MakePolygonalBaric`, `MakePolygonalLongedge`, and `MakepolygonalMedian` put the vertex in the middle point of the vertex-segment, in the barycenter of the endpoints of the incident edge-segments, on the endpoint of the longest incident edge-segment, on the median of the of the endpoints of the incident edge-segments, respectively.

A.3 Algorithmic Path `Lattice`

```
Lattice = multigraph > IsDirected > digraph > IsAcyclicDigraph > acyclicdigraph >
MakeSTDigraph > stdigraph > MakeSTPlanar > planarstdigraph > MakeReducedSTdigraph >
reducedplanarstdigraph > StraightLineDraw > StraightLine > DoPolygonal > polygonal
```

Note the `MakeSTPlanar` step, which is needed to convert the input graph to a planar graph and the `MakeReducedSTdigraph` step that breaks the transitive edges into two edges and a dummy intermediate vertex; such a dummy vertex will result in a bend of the final drawing.