DAG Drawing from an Information Visualization Perspective

G. Melançon and I. Herman

Centrum voor Wiskunde en Informatica (CWI)
P.O. Box 94079
1090 GB Amsterdam, The Netherlands
{Guy.Melancon,Ivan.Herman,}@cwi.nl

Abstract. When dealing with a graph, any visualization strategy must rely on a layout procedure at least to initiate the process. Because the visualization process evolves within an interactive environment the choice of this layout procedure is critical and will often be based on efficiency.
This paper compares two popular layout strategies, one based on the extraction of a spanning tree, the other based on edge crossing minimization of directed acyclic graphs. The comparison is made based on a large number of experimental evidence gathered through random graph generation. The main conclusion of these experiments is that, contrary to the popular belief, usage of edge crossing minimization algorithms may be extremely useful and advantageous, even under the heavy requirements of information visualization.

1 Introduction

Graph visualization has emerged lately as a sub–field of information visualization, specializing in visualizing data that comes with inherent relations. In that case, the data to be visualized is interpreted as a graph, and applications usually offer different strategies to view, to navigate in, and to interact with the graph. This has many areas of application in biology, chemistry, computer science, web navigation, or document management systems. See [6].

Graph visualization can rely on the rich body of knowledge developed over the years by the graph drawing community, which grew around the yearly Symposia on Graph Drawing. A large number of layout strategies are at disposal, which have been collected recently in the book of di Battista *et al.*[1]. However, it is not always easy to decide which layout algorithm to use for a given application and for a specific class of graphs: the practical requirements, concerns, assumptions, etc., are often different. The graph drawing community usually assumes graphs to be sparse and to contain only a small number of nodes. A few hundred nodes, containing only about two times more edges than the number of nodes, is a common measure. In contrast, the information visualization community has to deal with graphs often containing thousands of nodes. Of course, large graphs are often clustered, yielding much smaller graphs; nevertheless, the real size and density numbers are still different. This issue will be examined in more details later in the paper.

Trees occur quite often in information visualization, whenever the data form some kind of hierarchy. It is also generally admitted that tree layout algorithms have the lowest complexity. Trees are sparse graphs (trees with N nodes only have $N-1$ edges) and they can be efficiently traversed. Moreover, trees are planar and most algorithms will produce an "aesthetically pleasant" layout for a tree. The running time of a tree layout algorithm is usually low (see [1, Chap. 10]). As a consequence, tree visualization have a wide variety of usage in information visualization. The advantages of relying on tree drawing algorithms has motivated many authors to base their visualization strategy on the extraction and layout of a spanning tree [10, 11, 16] to gain the efficiency of the tree layout. Also, interacting with the graph often translates into the necessity of redrawing it a large number of times over very small time intervals; under those conditions, relying on the efficiency of a tree layout algorithms is sometimes viewed as a mandatory choice. One of the goals of this paper is to put this claim in a better perspective, and to show when it is justified and when it is not.

Directed acyclic graphs ("dag"–s) can be seen as a natural generalization of trees. They have no cycles and their nodes can be assigned to layers so that all edges point in the same direction, usually downwards. Many layout methods have been developed for these graphs, too. The core of most of these algorithms (usually referred to as the Sugiyama methods) mainly focus on minimizing the number of edge crossings on the generated layout. The reason is that a dag is generally not planar, and the usual assumption is that an "aesthetically pleasant" drawing of a dag is the one with a minimum number of edge crossings. The problem is that minimizing the number of crossings is difficult. The computation of the optimal solution compares to a classical sorting problem and is thus well above linear time complexity. However, some heuristics have been developed that produce a layout approaching the optimal solution.

Dag–s form a good intermediary class between trees and general graphs. Efficient methods exist which first extract a directed acyclic graph from a general graph (see [1, 3], for example), which makes them generally usable. As a consequence, instead of relying on spanning trees, information visualization systems could also decide to use the Sugiyama approach: extract a "spanning dag" first, layout the dag, and add the missing edges. The claims of "aesthetics", or "better readability" of the graph, which focuses on edge crossings may make this choice a good alternative.

The problem, which researchers in information visualization face, is that there aren't any systematic comparisons available in the literature, which would make it clear when the spanning tree approach is preferable over the spanning dag. As a consequence, most dag based algorithms are disregarded altogether, being viewed as too complicated anyway. Offering such a comparison is the main goal of this paper. The question addressed here is "under what limits is it reasonable to rely on a layout technique for directed acyclic graphs when visualizing a graph in an interactive environment?". The answer can be very valuable for information visualization: improving the overall readability of a graph layout is certainly desirable although this has to be balanced against response time and interactivity. Having a better knowledge on the limits of crossing minimization algorithms can only help to take the appropriate decision.

2 Algorithmic details

As we have said in the introduction, the goal is to compare two layout approaches: the first is based on spanning dag–s, whereas the second relies on the usage of spanning trees. Although both layout strategies concentrate on a special classes of graphs (dag–s and trees, respectively), their applicability is much wider. Many different algorithms extracting a spanning tree can be found in the literature[8] which makes the spanning tree approach fairly general. As for dag–s, several techniques exist, too, to extract an spanning acyclic subgraph. The reader should consult [1] for further details. Consequently, the methods being compared are both of a fairly general use.

2.1 Layout of dag–s

Global minimization of the number of crossings in a graph is a very difficult problem. Instead, a common approach is to look first at the more restricted problem of a graph consisting of two layers of nodes, where edges connect nodes in different layers only (also referred to as *bipartite* graphs). If the solution to this restricted problem is found, it can then be extended to the full graph by sweeping through the graph layer by layer. However, simply sweeping through the layers is not enough; one has to make use of the so–called *dummy nodes*, too. Indeed, in order to apply the Sugiyama algorithm, each edge has to be transformed into a sequence of appropriate edges only crossing neighbour levels. The dummy nodes (which are not visible on the final layout) are inserted between nodes which are not on neighbouring layers. Dummies are inserted as necessary to yield a series of truly bipartite graphs when sweeping through the layers.

Even the solution of the bipartite case has proven to be difficult, more exactly, to be NP-complete[4, 2]. Consequently, various heuristics have been explored to produce a sub–optimal solution (see, e.g., di Battista *et al.*[1, Section 9.2]). The layout strategy chosen for the purposes of this study is the so–called "barycentre" heuristic. The underlying idea is rather simple and is based on the intuition that a node should be positioned according to its neighbours' positions. Hence, given a node in a layer, the x coordinate of a node can be computed as being the barycentre of the coordinates of its neighbours. That process can be iterated to refine the final positions of nodes. Among all possible crossing minimization heuristics or algorithms, the barycentre heuristic possesses many attractive qualities that were stressed in a recent paper by Juenger and Mutzel[7]. Maybe the most important feature is that the complexity of the technique is *linear* in the number of nodes. Also, it provides a solution that compares very well to the optimal solution with respect to the number of crossings.

The linear complexity of the barycentre heuristics makes it a good competitor to the tree layout algorithm. The actual implementation developed for this study also minimizes the number of iterations further: assuming layer i has been assigned positions, we first compute the next layer of nodes at level $i + 1$. In doing this, we need to introduce new dummy nodes for every edge crossing level $i + 1$. The nodes on layer $i + 1$ are then positioned at the barycentric coordinate of their predecessors on layer i. The nodes on layer i are then revisited and positioned at the barycentric coordinate of *all their neighbours*, this time including those on layer $i + 1$. The implementation takes care of "throwing away" the dummy nodes which are no longer in use.

2.2 The spanning tree strategy

As said before, the book of Jungnickel[8] is an excellent source to find spanning tree extraction algorithms. A simple extraction satisfying no particular condition can be achieved within $|E|$ steps, where E denotes the set of edges in the graph. For example, a breath–first-search traversal of the graph with an appropriate selection criteria for edges will output a spanning tree of a graph. The extraction process can sometimes be bound to an optimization problem, too. Although it may be tempting to follow this connection in applications, the tree satisfying an optimization condition might have undesirable geometric properties, like being very poorly balanced. In other words, the extraction strategy must be carefully selected. The experiences leading to this paper have shown that iteratively building the spanning tree, selecting at each step a node reachable from the set of already selected nodes, is quite appropriate. The selection of the next node is made by minimizing the distance to the root (or starting point of the search). In most cases, this will result in a fairly balanced spanning trees.

The tree layout we used is the classical algorithm due to Reingold and Tilford [14, 15]. It runs in two steps, first traversing the tree in post–order and computing temporary coordinates and modifier fields values for each node. A second traversal, this time visiting the nodes in pre–order, computes the actual coordinates by cumulating the modifier values of nodes from the root to the leaves of the tree. This algorithm is linear in the number of nodes, in spite of the fact it also relies on the necessity of visiting and comparing coordinates of nodes on the right and left border of neighbor subtrees.

Care should be taken, however, when applying this algorithm to the spanning trees of dag–s. Indeed, the traditional Reingold and Tilford algorithm will position nodes at a specific depth in the tree on the same layer. However, the depth of a node in the spanning tree might not coincide with its level in the dag (and applications might require to keep this visual relation). One solution is to follow the same approach most implementations of the Sugiyama methods use, i.e., to introduce dummy nodes into the tree. This, however, may have very negative consequences on the memory and time requirements of the algorithm. Instead, in our case, the y-coordinate of a node is assigned according to its level in the dag, while preserving the x-coordinate as computed by the Reingold–Tilford algorithm. As a consequence, the load of introducing dummy nodes into the graph is avoided in this case.

Figure 4 in the appendix shows a tree with 200 nodes and about 400 edges laid out using the two different layout methods. We will comment on those in later sections.

3 The graph sample

The graphs, on which the experiments were based, were randomly generated using a Markov chain process. It would go beyond the scope of this paper to describe the details of how this Markov chain works, which will be published elsewhere [9]. Suffices it to say that, with no restriction, the chain produces a set of uniformly random dag–s (all dag–s appear with equal probability). It can be proven that the average edge density is of $N^2/4$ (note that this is higher than what the graph drawing community usually considers!). The interesting point is that the state space of the chain can be controlled by

imposing an upper bound on either the total number of edges in the dag, or the maximal degree of nodes. It appears that when restricting nodes to have a degree of at most k, the randomly generated graphs will have $kN/2$ edges on average. (The Markov chain works by systematically adding edges, on condition that a new edge does not create a cycle in the graph. The chain will tend to add edges until it reaches the maximum number of edges it can add. If all nodes reach degree k, the total number of edges must be $kN/2$.)

This Markov chain generation has proven to be a very useful tool because it offered a full control over the edge densities in the samples. With $k = 2$ (node degree at most 2), the average number of edges becomes N. Graphs with edge density N are obviously sparse since this condition will impose a large number of isolated vertices. When $k = 4$, the resulting sample consists of graphs which are considered as "dense" by the graph drawing community. However, in our view, $k = 4$ should not be considered as really dense for information visualization, and actually constitutes the lowest reasonable density to be examined (density $k = 4$ means that each data node has, on average, "relations" with at most 4 other data nodes, which is an unrealistic condition). Higher density cases have to be examined: graphs with edge density varying from $3N$ up to $5N$ (that is with k varying from 6 to 10) were also considered (although we only report here the cases with $k \leq 6$, see section 4).

For each value of $k = 2, \ldots, 10$, graphs of different sizes varying from 10 to 1000 nodes were generated. Exactly 20 graphs per category were processed 30 times through the layout algorithms, measuring the time it took to perform the layout itself (disregarding the time needed to display or read in the graph). Our conclusions are based on the average value of the results computed out of those 600 layout trials for each size category.

4 Results of the experimentation

We ran the experiment based on a Java implementation of the layout methods[1]; it can be expected that C implementations, for instance, would result in more efficient computing time. However, most of our conclusions are based on the relative efficiency measures of the two layout methods, i.e. our results remain valid in general, too. In the cases when we do refer to absolute timing results, our conclusions could only be made stronger by a faster implementation, as we will see in what follows.

4.1 Computing time complexity

As can be expected, time complexity is not a real concern for graphs with a few nodes and a low density (see Figures 1 & 2). The barycentre method is actually linear in the number of nodes (dummies *and* real nodes). With low density graphs, the number of dummies introduced in the layout process is low, as shown on Figure 3. For $k = 2, 3$ the number of dummies is at most equal to 4 or 5 times the number of real nodes in the graph. The curves in Figures 1 & 2 show that both layout techniques are equivalent with

[1] Our Java implementation is based on a general framework for graph visualization (see [5]).

respect to time in these cases; actually, the barycentre technique proves to be slightly superior when $k = 2$. Note also that for $k = 2$, even with graphs of size up to 1000 nodes, both algorithms have running time below half a second, which is acceptable when the layout occurs in an interactive environment. The same conclusion holds when $k = 3$, for graphs with up to 500 nodes.

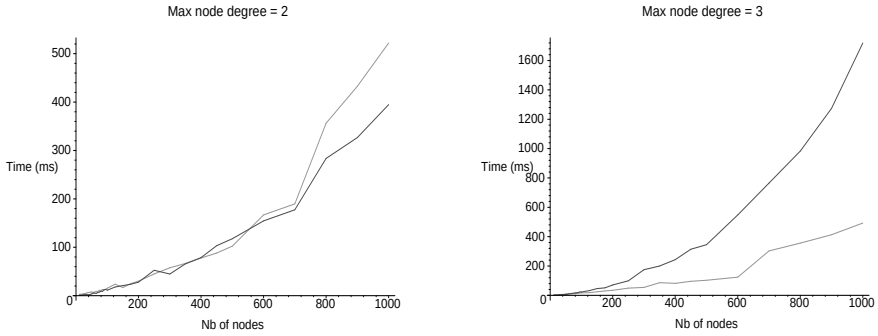

Fig. 1. Computing time for spanning tree and barycentre layouts. The gray colored curves show the spanning tree results, whereas the darker curves stand for the barycentre layout.

For graphs with higher densities, the number of inserted dummy nodes increases significantly. For a specific graph size, the number of dummy nodes appears to increase linearly with respect to edge density. Figure 3 shows that for a graph of size 250 the total number of nodes (dummies and real) varies from 5 to 30 times the actual number of real nodes, as k goes from 3 to 5. For graph of size 500, the same measure will vary from 5 to a little less than 50 times the number of nodes in the graph, over the same interval for edge density. These dummies create a significant load on the algorithm, because the weight of memory management will overshadow any theoretical speed improvement of the algorithm. Although the Java virtual machine might not be the best example for a good memory management, it is still significant that our implementation ran into serious memory fault problems at graphs with a few thousands nodes already. Hence, the barycentre approach seems to loose at higher densities; the extremely high number of dummies proves to be the Achilles' heel of the method.[2]

All these numbers seem to suggest that the barycentre method — or any other Sugiyama like layout method — is inappropriate for information visualization in real applications. However, one should not jump on conclusions too hastily. As Figures 1 & 2 show, for example, the barycentre method still scores well *in absolute times* for lower density graphs of size less than 400. When $k = 4$ (which we regard as the lowest

[2] This is, as a matter of fact, also an important conclusion of our measurements: researchers in graph drawing tend to concentrate on various ways of improving the edge crossing optimization step, and the problems incurred by the dummy nodes is often considered as less important.

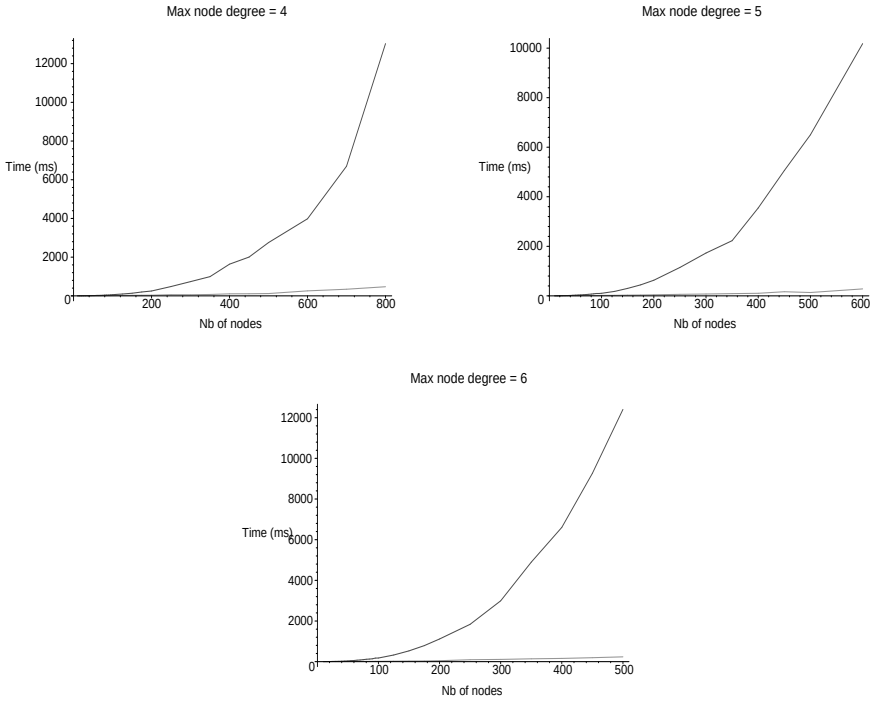

Fig. 2. Computing time for spanning tree and barycentre layouts (continued).

complexity level for information visualization) both the tree based and the barycentre strategies are comparable for graphs of a size up to about 250 nodes, since the barycentre method still lays out the graph in less than half a second. For higher densities, that is when $k = 5$ or 6, the same conclusion holds although the maximum graph size then has to be lowered down to remain under a 500 ms computing time (to 200 for $k = 5$ and to 150 for $k = 6$).[3] Although graphs with much more than 250 nodes frequently appear in applications, displaying all elements is often not a real option anyway: even the fastest and best layout strategy may produce a image barely understandable to humans. As a consequence, applications often offer an overview diagram of the graph (based, for example, on a suitable clustering of the original graph), and interaction occurs on this overview rather than the original data. Such an overview graph may have a size and edge density falling under the limits listed above, in which case it could be laid out using a Sugiyama method for improved readability.

[3] The reader should remember that these numbers stem from a Java implementation which could be improved, if necessary, by choosing a faster implementation environment.

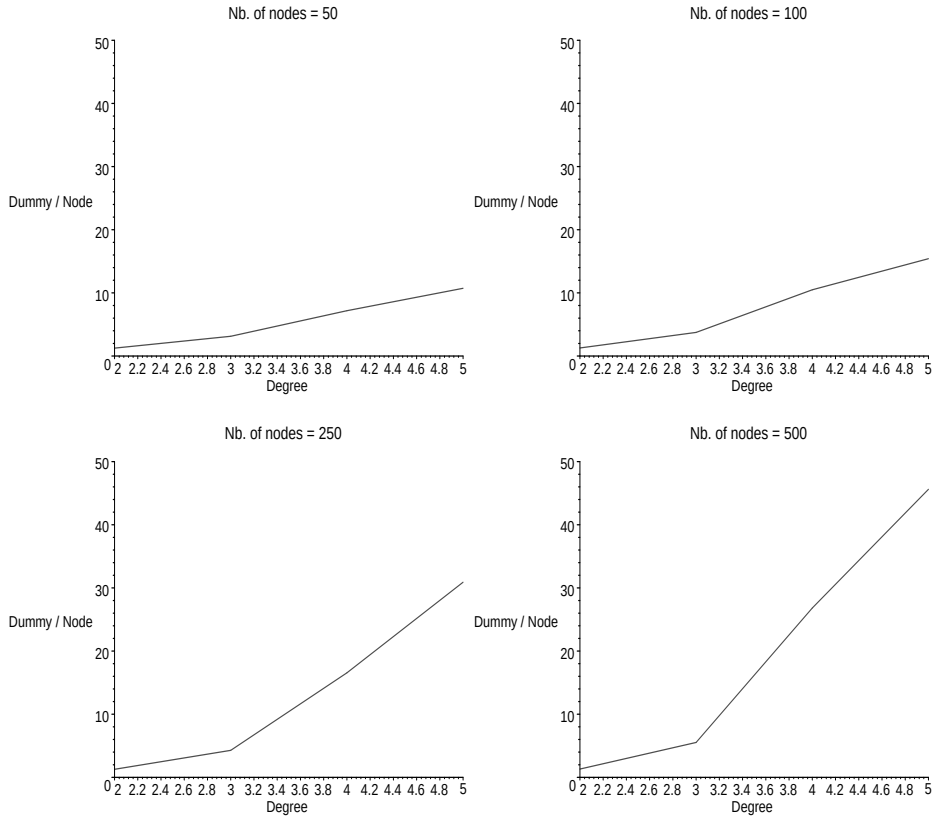

Fig. 3. Number of dummies introduced during barycentre layout (ratio per node)

4.2 Aesthetics

The usual reason for implementing the barycentre heuristics, or any other crossing minimization heuristics, is to produce a layout with as few edge crossings as possible. This aesthetics has been proven to have a measurable effect on the "readability" of the graph [12, 13], hence it may play a major role in the user interface aspect of the information visualization application. This, added to the fact that the barycentre method works in reasonable time when the size of the graph does not exceed some bounds, seems to make the barycentre method a real option.

There is a counter argument, though. One can show that as edge density increases, any crossing minimization algorithm will produce a solution that approaches the optimal solution (see [1, Section 9.2.5]). In other words, if the graph is dense, the number of unavoidable crossings will be high and any approach is susceptible to provide an acceptable solution — which will incidentally contain a large number of crossings. This argument supports the conclusion that the spanning tree strategy is as good as any other method, as far as edge crossings are concerned. So why bother?

The fact is that better analysis on the output of the barycentre algorithms reveals further qualities which may turn out to be important. Let us look at Figure 4. The barycentre layout results in less edge crossings, although this might not be immediately noticeable. However, the fact that the method positions a node at the barycentric x-coordinates of its neighbours explains why the overall picture has a more or less "slender" shape in the middle. The Reingold and Tilford layout is not affected in this way. The spanning tree extraction is completely separated from the positioning process and thus leads to a "square" image of the graph. The result is that nodes might be spread all over the displayable area, resulting in a large number of long, nearly horizontal edges: these are the edges which were disregarded by the spanning tree process. We suggest that the barycentre view, which tends to keep adjacent nodes together, gives a much better cognitive impression of the graph by offering a view closer to its "real" inherent structure.

Another important aspect in information visualization is the "predictability" of the layout when changes occur in the graph. What this means is that small changes in the graph should not result in significant layout changes; this is extremely important if the graph is interacted with. A layout strategy based on a spanning tree extraction process might not fulfill this requirement: the spanning tree extraction could result in a radically different tree even if only one single node or edge is, for example, added to the graph. Hence the graph layout will radically change. This is not the case of the barycentre method, where any effect of interaction will remain local. Although it is certainly possible to carefully design the tree extraction process and preserve predictability, this intrinsic quality of the barycentre method can be a decisive reason for choosing it.

5 Conclusion and Further Research

Our experiments with the two layout strategies have shown that the popular belief telling us that information visualization should rely on spanning tree algorithms is not really correct. Edge crossing minimization methods for dag–s should not be disregarded, as they often are: with very reasonable restrictions on the size and the densities of the graphs, these methods do not only compete with tree based layouts in terms of time, but the quality of the output they produce may be superior.

Edge crossing minimization is not the only graph layout strategy which has been developed over the years. We think that a systematic overview and comparison of other methods (like the force–directed ones, for example), confronted with the requirements of information visualization, is necessary. Such comparisons would help the information visualization community to choose the right approach when implementing a particular application. Care should be taken to base these comparisons on real data and implementations, rather than on theoretical results only; our measurement results on the explosion on dummy nodes is a good example for the problems which can be revealed that way.

References

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

2. Eades, P. and Whitesides, S.H. Drawing Graphs in Two Layers. *Theoretical Computer Science*, 131(2):361–374, 1994.
3. Gansner, E.R., North, S.C., and Vo, K.P. DAG - A Program that Draws Directed Graphs. *Software: Practice and Experience*, 18(11):1047–1062, 1988.
4. Garey, M.R. and Johnson, D.S. Crossing number is NP-complete. *SIAM Journal of Algebraic and Discrete Methods*, 4(3):312–316, 1983.
5. Herman I., Marshall M. S., and Melançon G. An Object-Oriented Design for Graph Visualization. Technical Report INS-0001, Centrum voor Wiskunde en Informatica (CWI), 2000. See `ftp://ftp/cwi/nl/pub/CWIreports/INS`.
6. Herman I., Marshall M. S., and Melançon G. Graph Visualization and Navigation in Information Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 6, 2000 (to appear).
7. Juenger, M. and Mutzel, P. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications*, 1(25):33–59, 1997.
8. Jungnickel, D. *Graphs, Networks and Algorithms*. Springer, 1999.
9. Melançon G., Dutour I., and Bousquet-Mélou M. Random generation of Dags for Graph Drawing. Technical Report INS-0005, Centrum voor Wiskunde en Informatica (CWI), 2000. See `ftp://ftp/cwi/nl/pub/CWIreports/INS`.
10. Munzner, T. H3: Laying out Large Directed Graphs in 3D Hyperbolic Space. In *IEEE Symposium on Information Visualization (InfoVis '97)*, pages 2–10. IEEE CS Press, 1997.
11. Munzner, T. Drawing Large Graphs with H3Viewer and Site Manager. In *Symposium on Graph Drawing GD '98*, Lecture Notes in Computer Science, pages 384–393. Springer-Verlag, 1998.
12. Purchase, H., Cohen, R.F., and James, M. Validating Graph Drawing Aesthetics. In *Symposium Graph Drawing GD'95*, volume 1027 of *Lectures Notes in Computer Science*, pages 435–446. Springer-Verlag, 1995.
13. Purchase, H.C. Which aesthetic has the greatest effect on human understanding? In *Symposium on Graph Drawing GD '97*, Lecture Notes in Computer Science, pages 248–261. Springer-Verlag, 1998.
14. Reingold, E.M. and Tilford, J.S. Tidier Drawing of Trees. *IEEE Transactions on Software Engineering*, 7(2):223–228, 1981.
15. Walker, J.Q. A Node-positioning Algorithm for General Trees. *Software: Practice and Experience*, 20(7):685–705, 1990.
16. Wills, G.J. Niche Works - Interactive Visualization of Very Large Graphs. In *Symposium on Graph Drawing GD'97*, volume 1353 of *Lectures Notes in Computer Science*, pages 403–414. Springer, 1997.

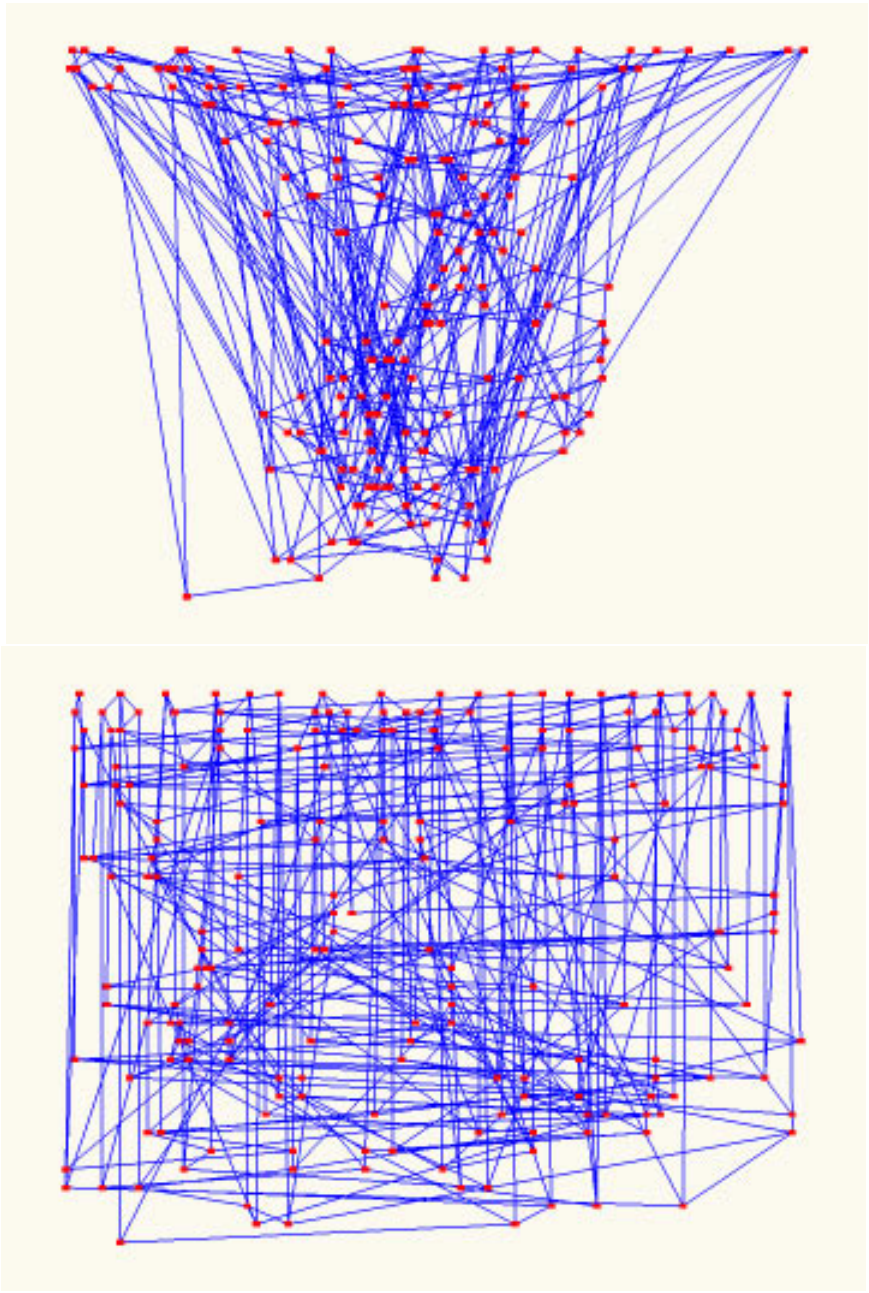

Fig. 4. Layout examples. Barycenter method (above) and spanning tree layout adapted from Reingold and Tilford (below).