Flexible Layering in Hierarchical Drawings with Nodes of Arbitrary Size

Carsten Friedrich[1] **Falk Schreiber**[2]

[1]Capital Markets CRC, Sydney, NSW 2006, Australia
Email: cfriedrich@cmcrc.com

[2]Bioinformatics Center, Institute of Plant Genetics and Crop Plant Research,
Corrensstr. 3, 06466 Gatersleben, Germany
Email: schreibe@ipk-gatersleben.de

Abstract

Graph drawing is an important area of information visualization which concerns itself with the visualization of relational data structures. Relational data like networks, hierarchies, or database schemas can be modelled by graphs and represented visually using graph drawing algorithms. Most existing graph drawing algorithms do not consider the size of nodes when creating a drawing. In most real world applications, however, nodes contain information which has to be displayed and nodes thus need a specific area to display this information. The required area can vary significantly between different nodes in the same graph. In this paper we present an algorithm for the layering step of hierarchical graph drawing methods that is able to take the sizes of the nodes into account. It further allows the user to choose between compact drawings with many temporary (dummy) nodes and less compact drawings with fewer dummy nodes. A large number of dummy nodes can significantly increase the running time of the subsequent steps of hierarchical graph drawing methods.

Keywords: Graph drawing, Graph visualization, Layering

1 Introduction

Most algorithms for drawing graphs do not consider the size of nodes when creating a drawing and treat them as points (for example in (Eades 1984) for force-directed layouts, in (Sugiyama, Tagawa & Toda 1981) for hierarchical layouts, and in (Tamassia 1987) for orthogonal layouts). For real world graph drawing applications, however, it is almost always essential to be able to account for different node sizes when generating a drawing. Examples include the visualization of UML class diagrams, where nodes can have different sizes depending on the number of attributes and methods, the drawing of program flow graphs with blocks of different sizes, or the visualization of biochemical reaction networks where the node sizes depend on the underlying biochemical reaction (Fig. 1).

Standard hierarchical graph drawing algorithms consist of three phases (Sugiyama et al. 1981):

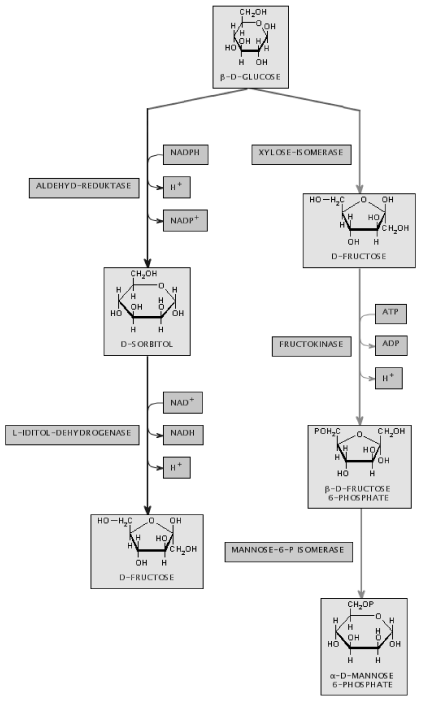
1. The assignment of nodes to layers (*layering*),
2. The permutation of nodes within layers (*crossing reduction*), and
3. The assignment of coordinates to nodes and bend points of edges (*coordinate assignment*).

If node sizes are not taken into account during this process, the quality of the layout is usually significantly reduced when the nodes and edges are actually drawn, as nodes and edges may suddenly overlap, although they did not from the algorithm's point of view. If the drawing is scaled to avoid this effect the drawing usually loses compactness, thus violating another important criteria for a good graph drawing (see (Di Battista, Eades, Tamassia & Tollis 1999) and Fig. 2 and 3. Note that Fig. 2(b), where the chain of small nodes is placed beside the large nodes, shows a more compact drawing of the same graph as in Fig. 2(a)).

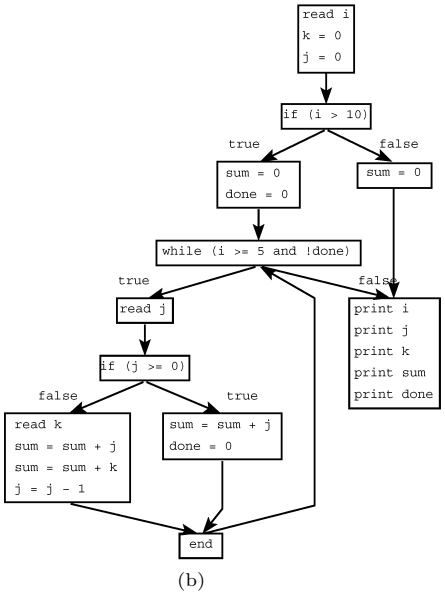
It is easy to consider the width of nodes in hierarchical drawings and this is therefore implemented in many algorithms (Gansner, Koutsofios, North & Vo 1993, Koutsofios & North 1993, Messinger, Rowe & Henry 1991, North & Woodhull 2002, Rowe, Davis, Messinger, Meyer, Spirakis & Tuan 1987, Sander 1999). No simple and good methods for considering the exact heights of nodes in hierarchical drawings are known, however. Consequently, most methods which consider node heights at all only apply very naïve heuristics, for example assigning the maximum node height to all nodes. These naïve heuristics often lead to unnecessarily large drawings, as can be seen, for example, in Fig. 4(b). North and Woodhull introduce a more sophisticated heuristic in (North & Woodhull 2002). A discussion of their approach in respect to the method presented in this paper can be found in Sec. 3.

In this paper, we present an algorithm for the layering step of hierarchical graph drawing algorithms which takes the height of nodes into account. The algorithm has an inherent trade-off between the area of the drawing and the number of additional dummy nodes. A large number of dummy nodes can significantly increase the running time of the subsequent steps of hierarchical graph drawing methods. Therefore our algorithm allows the user to choose between compact layouts with many dummy nodes during the subsequent steps or less compact layouts with fewer dummy nodes.

The paper is structured as follows: In Sec. 2 we discuss possible approaches to considering different node sizes in graph drawings. Some terminology and the special case of handling node sizes in hierarchical graph drawing algorithms are described in Sec. 3. Our new method is presented in Sec. 4. Finally, in Sec. 5 we give some experimental results.

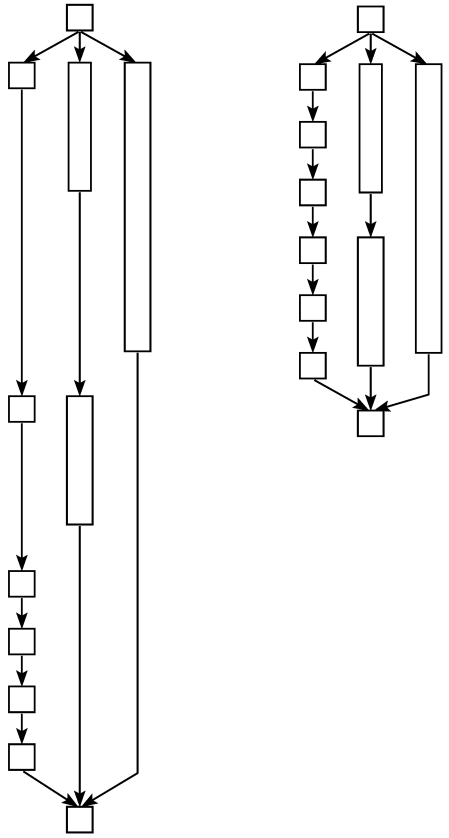


(a)



(b)

Figure 1: Two application examples: Visualization of (a) a small biochemical reaction network and (b) a program flow diagram.



(a) (b)

Figure 2: Two hierarchical drawings of the same graph: (a) using a standard hierarchical graph drawing algorithm (Sugiyama et al. 1981), (b) a more compact drawing.

2 General approaches to considering node sizes

There are three main approaches to considering node sizes in graph drawing methods:

1. Enlarging all nodes to the size of the largest node. Subsequently the modified graph is drawn using standard algorithms for the layout of graphs with uniform node sizes. Early graph drawing methods (Batini, Furlani & Nardelli 1985) followed this idea.

2. Post-editing the drawing. First, the nodes are considered as being points and placed by a standard graph drawing algorithm. In the second step they are enlarged to their original size. Finally, the drawing is modified in such a way that all node intersections are removed while preserving the relative node positions (Misue, Eades, Lai & Sugiyama 1995) (Fig. 3(a)-(c)).

   This approach often produces visualizations which do not respect important aesthetic criteria, such as minimizing edge-node crossings (see Fig. 3(c)). A similar method where nodes are enlarged to the size of the largest node; placed by a graph drawing algorithm; scaled down to their original size; and finally, rearranged in a more compact way while preserving the mental map has the same disadvantage.

3. Considering the real size of each node using improved or new graph drawing algorithms. There are specialized solutions for some classes

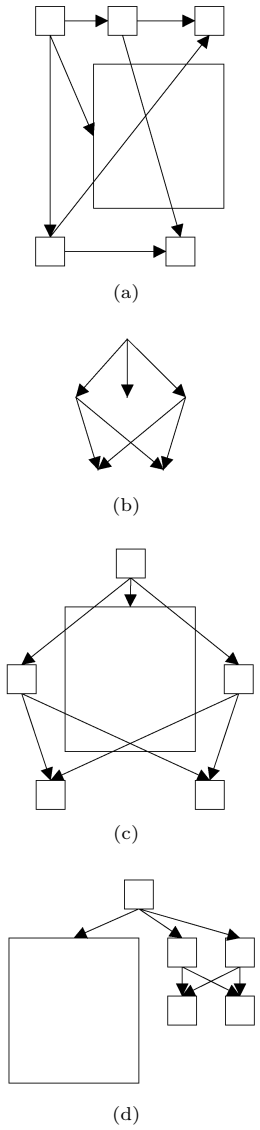


Figure 3: Post-editing of a drawing. (a) Initial drawing, (b) reduction of the node sizes to points and placement by a hierarchical layout algorithm, (c) enlargement of the nodes to their original sizes and mental map preserving rearrangement. This drawing contains node-edge-crossings and is taller than necessary. (d) A better drawing of the same graph.

of layout algorithms, such as orthogonal layouts (Biedl, Madden & Tollis 1997, Di Battista, Didimo, Patrignani & Pizzonia 1999, Fößmeier & Kaufmann 1997), drawings of trees (Bloesch 1993), and force directed methods (Gansner & North 1998). Existing solutions for hierarchical layouts are discussed in Sec. 3.

3 Considering node sizes in hierarchical layouts

We consider directed, connected, acyclic graphs $G = (V, E)$. Let $pred(v)$ be the set of predecessors and $ind(v)$ be the number of incoming edges of a node $v \in V$. A *layering* $(L_1, ..., L_l)$ of G is an ordered partition of V into non-empty layers such that adjacent nodes are in different layers. $L_i = (v_1, \ldots, v_k)$ is the *i-th layer* of G and $L(v) = i$ if $v \in L_i$. The *span* of an edge $(u, v) \in E$ is given by $S((u, v)) = L(v) - L(u)$. A layering is called *simple* if the span of all edges is one.

In hierarchical graph drawings the nodes are placed on parallel lines corresponding to the layering $(L_1, ..., L_l)$. Without loss of generality, the hierarchy is drawn top-to-bottom and the layers are horizontal lines. In a drawing of a graph, each node v is represented by a rectangle with height $h(v)$ and width $w(v)$. The center of the rectangle is given by $(x(v), y(v))$. The upper boundary of the rectangle $y(v) - h(v)/2$ is denoted by y_u. The lower boundary $y(v) + h(v)/2$ is denoted by y_l. Each edge e is represented by a polyline $p(e) = (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ where $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ are the bends of the polyline.

A layering is associated with the y-coordinates of nodes such that usually all nodes of a layer have the same y_u-coordinate. There are different ways to compute the y-coordinates of nodes in hierarchical graph drawings in the presence of large nodes:

Definition 1 *Given a graph $G = (V, E)$ and the minimum distance between nodes m_d, a layering of G is called:*

1. size-free,
   if $\forall (u, v) \in E : y(u) + m_d \leq y(v)$ *(Fig. 4(a))*
2. maximal,
   if $\forall (u, v) \in E : y_u(u) + h_{max} + m_d \leq y_u(v)$
   with $h_{max} = max\{h(w) \mid w \in V\}$ *(Fig. 4(b))*
3. layer-maximal,
   if $\forall (u, v) \in E : y_u(u) + h_l + m_d \leq y_u(v)$
   with $h_l = max\{h(w) \mid w \in V, y_u(u) = y_u(w)\}$[1] *(Fig. 4(c))*
4. size-true,
   if $\forall (u, v) \in E : y_u(u) + h(u) + m_d \leq y_u(v)$ *(Fig. 4(d))*

A layering is called minimal *if* $max\{y_l(v) \mid v \in V\}$ *is minimal and* $\forall v \in V : y_u(v) \geq 0$.

Common algorithms for hierarchical graph drawings do not consider node sizes (*size-free* layering) (Sugiyama et al. 1981), or compute *maximal* (Carpano 1980, Koutsofios & North 1993, Rowe et al. 1987), or *layer-maximal* layerings (Gansner et al. 1993, Messinger et al. 1991, Sander 1999). If the node sizes are given, these three layerings can be easily transformed into each other. We will call these layering approaches *global layering*. Global layering of graphs tends to produce large drawings. In contrast, *size-true* layering leads to compact drawings, as can be seen by comparing Fig. 4(b,c) with Fig. 4(d). In the latter case, the assignment of nodes to layers depends not only on the topological sorting of G, but also on the individual height of each node. We call this solution *local layering*. North and Woodhull presented an algorithm for computing size-true layerings (North & Woodhull 2002) where a new layer is introduced for each y_u and y_l. In a subsequent step nodes and edges crossing one or more layers are split into chains of nodes to obtain a simple layering.

Local layering can lead to new problems:

1. Crossings between nodes and edges can be unavoidable, as can be seen in Fig. 5. However, even with such crossings, a compact visualization may be easier to understand than a conventional hierarchical drawing because the short edges in the compact drawing are easier to follow than long edges.
2. The minimum distance between nodes can no longer be guaranteed by an additional distance between layers. To avoid problems when placing a node, it is therefore necessary to consider the

[1] The variable h_l is the height of the largest node in the same layer as node u.

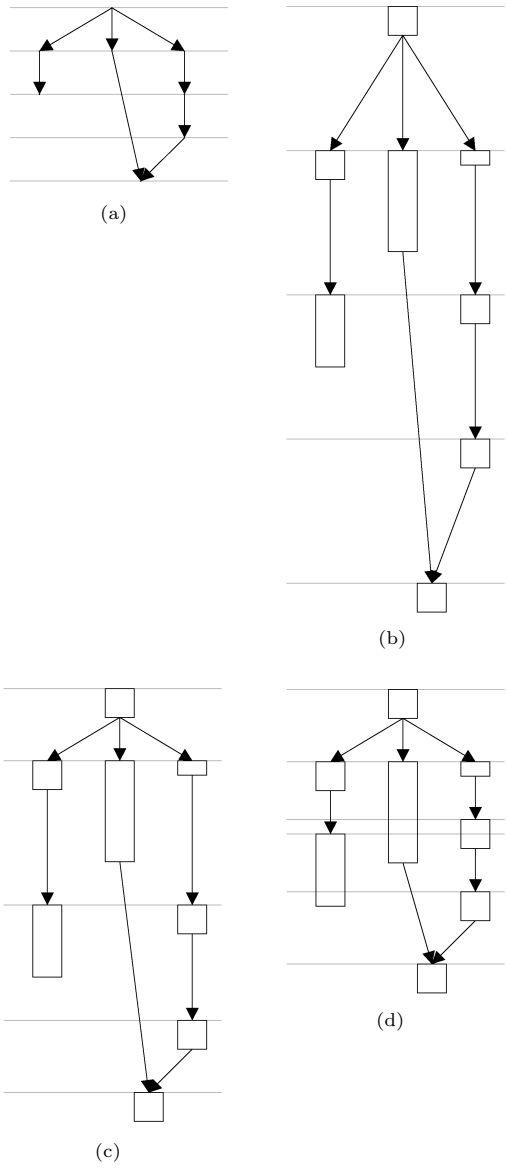


Figure 4: Different approaches to place nodes into layers and to compute their y-coordinates. While the number of layers is the same for (a)-(c), it is often higher for *size-true* placement (d).

minimum distance m_d to all nodes of the previous layer. This can be done by adding m_d to the height of each node. Nodes which could otherwise violate the minimum distance are thus enlarged into the next layer and consequently represented by a dummy node in that layer. The dummy node prevents other nodes from occupying the space underneath the large node.

3. Local layering can result in many additional layers and therefore many dummy nodes (see Fig. 6). Even if nodes differ only slightly in their $y_{u/l}$-coordinates this introduces two new layers for each node. Note that in Fig. 6 only new layers for y_u-coordinates are shown. Many layers have a negative effect on the running time of the subsequent steps (crossing reduction, computation of x-coordinates). Therefore the number of layers should be as small as possible.

The layering step in (North & Woodhull 2002) produces up to two layers for each node. A solution where only y_u values result in layers (as shown in Fig. 4(d) and Fig. 6) reduces the number of layers, but can introduce unnecessary crossings and therefore needs care during the edge routing or a special edge routing step, e. g. (Dobkin, Gansner, Koutsofios & North 1997). However, even this solution usually results in many layers and a huge number of dummy nodes. For sparse graphs with 200 nodes, 3000 edges, and random height of nodes between 1 and 100 an average of more than 11000 dummy nodes are inserted into the graph (Fig. 9). In contrast, there is an average of less than 2000 dummy nodes for these graphs using *layer-maximal* layering.

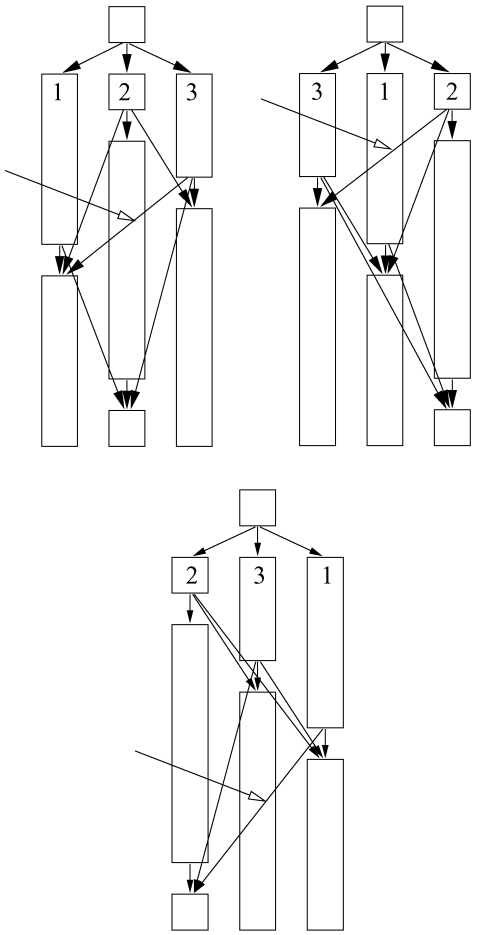


Figure 5: Three different drawings of the same graph. They represent all essential permutations of nodes (Note, that the drawing for the node order 1-2-3 has the same unavoidable node-edge crossing as the drawing for 3-2-1.). The arrows with white points refer to unavoidable crossings between nodes and edges in these compact drawings. Note, that hierarchical graph drawing algorithms draw edges strictly downwards, therefore it is not possible to avoid the marked node-edge crossings by routing the edges around the nodes.

Compact drawings are desirable. However, the existing solutions for local layering yield a huge number of dummy nodes and therefore long running times in the subsequent steps. This motivates the development of an algorithm with local layering, where the user can choose between compact layouts with many dummy nodes and less compact layouts with fewer dummy nodes.

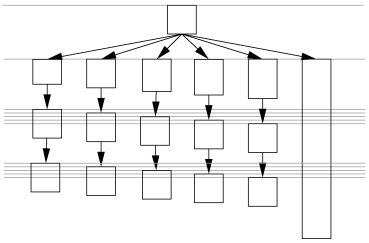


Figure 6: Local layering can result in many additional layers and therefore many dummy nodes.

4 Algorithm

Our algorithm is based on the following idea. The layers are computed from top to bottom. The y_u-coordinate of a node and its layer are computed together. Each node is placed only once. During the placement of nodes in layer L_i all possible subsequent layers are joined into one layer, if they are situated in a predefined horizontal area. The height of this horizontal area is given by l_{height}.

The main loop of the algorithm works as follows (Fig. 7 and Fig. 8): We assume that we have already placed the nodes of the layers L_1 to L_{i-1} and determined the y-coordinates of these layers. We further assume that we have determined all nodes which should be placed on layer L_i, as well as the y-coordinate of layer L_i. Now we do four things:

1. compute the y-coordinate of the next layer L_{i+1},
2. place all nodes on layer L_i,
3. split nodes which are too tall into two nodes on layer L_i and L_{i+1}, and
4. determine the nodes which have to be placed on layer L_{i+1}.

Figure 7 illustrates these steps. Figure 7(a) shows a part of a graph, the three top nodes have to be placed on layer L_i. Figure 7(b) displays the start situation; no node is placed. Unplaced nodes are drawn with dashed lines. Nodes are displayed with their original height and the additional minimum distance m_d in grey.

We use a heap (priority queue) H to store all nodes of the current layer L_i in increasing height. The smallest node, v, is used to compute the minimal y-coordinate for the next layer: $y_{min} = y_u(v) + h(v)$. The maximal y-coordinate for the next layer is given by $y_{max} = y_{min} + l_{height}$. Figure 7(c) shows the area between y_{min} and y_{max} in light grey. All nodes which end in this area are placed completely on layer L_i and deleted from H (Fig. 7(d)). The y-coordinate of layer L_{i+1}, y_{i+1}, depends on the maximum height of these nodes, as shown in Fig. 7(e). Each of the remaining nodes $v \in H$ is too large and thus split into two nodes v_1, v_2 and a connecting edge. Node v_1 is placed on layer L_i, $h(v_1) = y_{i+1} - y_i, h(v_2) = h(v) - h(v_1)$. The node v_2 replaces v in the heap H. Note that the height of all nodes in H is reduced by the same amount, therefore the order of nodes in the heap is unchanged. Now all nodes of layer L_i are placed.

To place the nodes of the next layer L_{i+1}, we add to the heap all those nodes which are not in H and have not yet been placed but all of whose predecessors have been placed. Then we run the loop again for layer L_{i+1} (Fig. 7(f)). The complete algorithm is shown in Fig. 8. The set V_{placed} keeps track of which nodes are already placed and $L(v)$ stores the layer for each node v. The minimum distance m_d is added to the nodes before the algorithm.

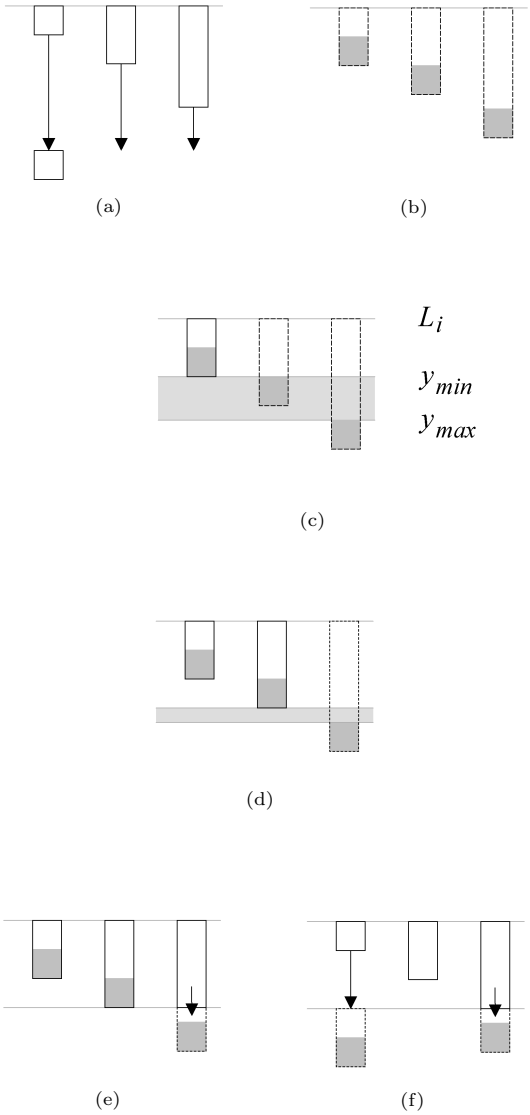


Figure 7: Placing of nodes of layer L_i.

Theorem 1 *The algorithm computes a layering in* $O(|V|^2)$.

Proof The *forall* loop (lines 1-4) is bound by $O(|V| + |E|)$ and the build of the heap in line 6 takes $O(|V| \log |V|)$.

The *while* loop (lines 11-49) runs at most $O(|V|)$ times because up to $|V|$ layers can be created (see Fig. 6 for an example).

There is a maximum of $|V|$ nodes per layer, including the nodes which might be created by splitting larger nodes. The inner *while* loop (lines 24 to 31) runs at most $|V|$ times. The inner *forall* loop (lines 35 to 44) also runs at most $|V|$ times as splitting a node and updating the various data structures can be done in constant time. Decreasing the counter in the children of placed nodes does not add to the overall runtime, as each edge in the graph is visited only once and the overall runtime of decreasing counter is therefore bound by $O(|V|^2)$.

```
1  forall (v ∈ V)
2     Initialize counter in v
3     with number of incoming edges.
4  end forall
5
6  Insert {v | v ∈ V, ind(v) = 0} into heap H;
7  y_next := 0;
8  curLayer := 0;
9  V_placed = ∅;
10
11 while (H ≠ ∅)
12    // Place all nodes which are completely on
13       the current layer
14    curLayer := curLayer + 1;
15    y_cur := y_next;
16    v := delMin(H);
17    L(v) := curLayer;
18    y_u(v) := y_cur;
19    V_placed := V_placed ∪ {v};
20    y_min := y_cur + h(v);
21    y_max := y_min + l_height;
22    y_next := y_min;
23
24    while (y_cur + h(min(H)) ≤ y_max)
25       v := delMin(H);
26       L(v) := curLayer;
27       y_u(v) := y_cur;
28       forall (u ∈ children(v))
29          decrease counter in u
30       end forall
31    end while
32    y_next := y_cur + h(v);
33
34    // Split large nodes (on this and next layer)
35    forall (v ∈ H)
36       In G = (V, E) replace v by v_1, v_2 and
37          the edge (v_1, v_2);
38       V_placed := (V_placed \ {v}) ∪ {v_1};
39       L(v_1) := curLayer;
40       y_u(v_1) := y_cur;
41       h(v_1) := y_next − y_cur;
42       h(v_2) := h(v) − h(v_1);
43       Replace in heap H node v by node v_2;
44    end forall
45
46    // Compute other nodes of the next layer
47    Insert {v | v ∈ V, v ∉ H, v ∉ V_placed,
48       counter(v) = 0} into heap H;
49 end while
```

Figure 8: Algorithm: Layering

We can determine which nodes we have to add to the heap (lines 47/48) in $O(|V|)$ by adding all those nodes whose counter is 0 and which are not placed yet. Adding a node to the heap takes $O(\log|V|)$. As each node has to be added to the heap only once during the execution of the algorithm (as nodes created by splitting are already in the correct order in the heap) the addition of nodes to the heap does not add to the overall runtime of $O(|V|^2)$. ◇

The final part of the layering step of hierarchical graph drawing algorithms is the replacing of each edge-layer crossings by a dummy node in order to compute a so called *simple* layering (not shown in the algorithm). Note that this step takes $O(|V| * |E|)$ in both, the conventional layering approaches and our method. $O(|V| * |E|)$ is also an upper bound for the number of nodes in the graph $G' = (V', E')$ after the layering algorithm, therefore our new algorithm does not change the overall complexity of the layering step of the hierarchical graph drawing algorithms.

5 Discussion

Our algorithm can be used to compute a wide range of different layerings (including the conventional solutions) depending on the value of the parameter l_{height}.

1. $l_{height} = 0$ gives a minimal *size-true* layering (Fig. 11(a)),
2. $0 \leq l_{height} \leq max\{h(v) \,|\, v \in V\} + m_d$ gives less compact but dummy node reduced layerings (Fig. 11(b)), and
3. $l_{height} = max\{h(v) \,|\, v \in V\} + m_d$ gives a *layer maximal* layering (Fig 11(c)).
4. A *maximal* layering can be achieved with $l_{height} = max\{h(v) \,|\, v \in V\} + m_d$ and a change of line 32 to $y_{next} := y_{cur} + l_{height}$.

The parameter l_{height} has a strong influence on the height of the drawing and the computation time of the subsequent steps of the graph drawing algorithm. A small value of l_{height} results in many layers (that is, many dummy nodes and thus long computation times for the next steps) but a compact layout. A high value of l_{height} yields few layers (that is, few dummy nodes and thus short computation times for the next steps) but also a less compact drawing. Figures 9 and 10 show some results for example graphs. We notice a high number of dummy nodes for very small values of l_{height} (Fig. 9). However, the height of the drawing increases only slowly with increasing value of l_{height} as shown in Fig. 10. Therefore a small value for l_{height} seems to be a good compromise between the compactness of the drawing and the number of dummy nodes.

Note that using this layering algorithm requires special care during the crossing reduction and the coordinate assignment. The crossing reduction has to guarantee that there are no crossings between parts of large (split) nodes. Additionally, the coordinate assignment must assign the same x-coordinate to all parts of a large node.

This layering algorithm has been implemented in the Graphlet system (Himsolt 2000) and is used in BioPath (Forster, Pick, Raitner, Schreiber & Brandenburg 2002), a system for interactive visualization of biochemical pathways.

Acknowledgements

This work was supported by the German Ministry of Education and Research (BMBF) under grant 0312706A and the Australian Research Council.

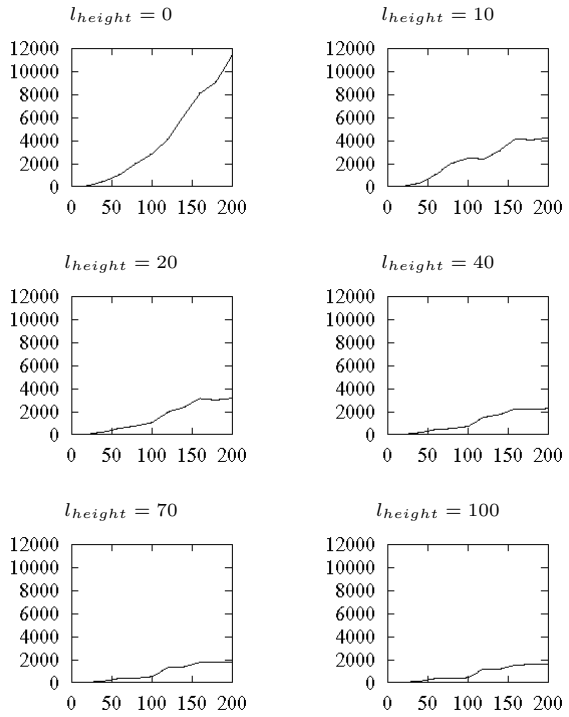


Figure 9: Number of dummy nodes (y-axis) depending on the value of l_{height}. The original graphs have 10, 20,..., 200 nodes (x-axis), density 0.15, and random height of nodes between 1 and 100. The diagrams show the average over 2000 graphs.

References

Batini, C., Furlani, L. & Nardelli, E. (1985), What is a Good Diagram? A Pragmatic Approach, *in* 'Proc. of the 4th Intl. Conf. on Entity-Relationship Approach', pp. 312–319.

Biedl, T. C., Madden, B. P. & Tollis, I. G. (1997), The Three-Phase Method: A Unified Approach to Orthogonal Graph Drawing, *in* G. Di Battista, ed., 'Proc. of the 5th Intl. Symp. on Graph Drawing (GD'97)', Vol. 1353 of *LNCS*, Springer-Verlag, pp. 391–402.

Bloesch, A. (1993), 'Aesthetic Layout of Generalized Trees', *Software - Practice and Experience* **23**(8), 817–827.

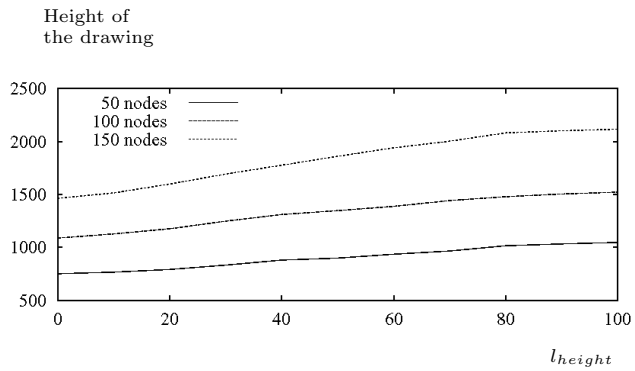


Figure 10: The height of the drawing depending on the value of l_{height} for graphs with 50, 100, and 150 nodes as in Fig. 9.

Carpano, M.-J. (1980), 'Automatic Display for Hierarchized Graphs for Computer-Aided Decision Analysis', *IEEE Trans. on Systems, Man, and Cybernetics* **10**(11), 705–715.

Di Battista, G., Didimo, W., Patrignani, M. & Pizzonia, M. (1999), Orthogonal and Quasi-Upward Drawings with Vertices of Arbitrary Size, *in* J. Kratochvil, ed., 'Proc. of the 7th Intl. Symp. on Graph Drawing (GD'99)', Vol. 1731 of *LNCS*, Springer-Verlag, pp. 297–310.

Di Battista, G., Eades, P., Tamassia, R. & Tollis, I. G. (1999), *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, New Jersey.

Dobkin, D. P., Gansner, E. R., Koutsofios, E. & North, S. C. (1997), Implementing a General-Purpose Edge Router, *in* G. Di Battista, ed., 'Proc. of the 5th Intl. Symp. on Graph Drawing (GD'97)', Vol. 1353 of *LNCS*, Springer-Verlag, pp. 262–271.

Eades, P. (1984), 'A Heuristic for Graph Drawing', *Congressus Numerantium* **42**, 149–160.

Forster, M., Pick, A., Raitner, M., Schreiber, F. & Brandenburg, F. J. (2002), 'The system architecture of the BioPath system', *In Silico Biology* **2**(3), 415–426.

Fößmeier, U. & Kaufmann, M. (1997), Algorithms and Area Bounds for Nonplanar Orthogonal Drawings, *in* G. Di Battista, ed., 'Proc. of the 5th Intl. Symp. on Graph Drawing (GD'97)', Vol. 1353 of *LNCS*, Springer-Verlag, pp. 134–145.

Gansner, E. R., Koutsofios, E., North, S. C. & Vo, K. P. (1993), 'A Technique for Drawing Directed Graphs', *IEEE Trans. on Software Engineering* **19**(3), 214–230.

Gansner, E. R. & North, S. C. (1998), Improved Force-Directed Layouts, *in* S. H. Whitesides, ed., 'Proc. of the 6th Intl. Symp. on Graph Drawing (GD'98)', Vol. 1547 of *LNCS*, Springer-Verlag, pp. 364–373.

Himsolt, M. (2000), 'Graphlet: Design and Implementation of a Graph Editor', *Software - Practice and Experience* **30**(11), 1303–1324.

Koutsofios, E. & North, S. C. (1993), Drawing Graphs with Dot, Technical report, AT&T Bell Laboratories, Murray Hill NJ.

Messinger, E. B., Rowe, L. A. & Henry, R. R. (1991), 'A Divide-and-Conquer Algorithm for the Automatic Layout of Large Directed Graphs', *IEEE Trans. on Systems, Man, and Cybernetics* **21**(1), 1–11.

Misue, K., Eades, P., Lai, W. & Sugiyama, K. (1995), 'Layout Adjustment and the Mental Map', *Journal of Visual Languages and Computing* **6**, 183–210.

North, S. & Woodhull, G. (2002), Online Hierarchical Graph Drawing, *in* P. Mutzel et al., eds., 'Proc. of the 9th Intl. Symp. on Graph Drawing (GD 2001)', Vol. 2265 of *LNCS*, Springer-Verlag, pp. 232–246.

Rowe, L. A., Davis, M., Messinger, E., Meyer, C., Spirakis, C. & Tuan, A. (1987), 'A Browser for Directed Graphs', *Software - Practice and Experience* **17**(1), 61–76.

Sander, G. (1999), 'Graph Layout for Applications in Compiler Construction', *Theoretical Computer Science* **217**(2), 175–214.

Sugiyama, K., Tagawa, S. & Toda, M. (1981), 'Methods for Visual Understanding of Hierarchical System Structures', *IEEE Trans. on Systems, Man, and Cybernetics* **SMC-11**(2), 109–125.

Tamassia, R. (1987), 'On Embedding a Graph in the Grid with the Minimum Number of Bends', *SIAM Journal on Computing* **16**(3), 421–444.

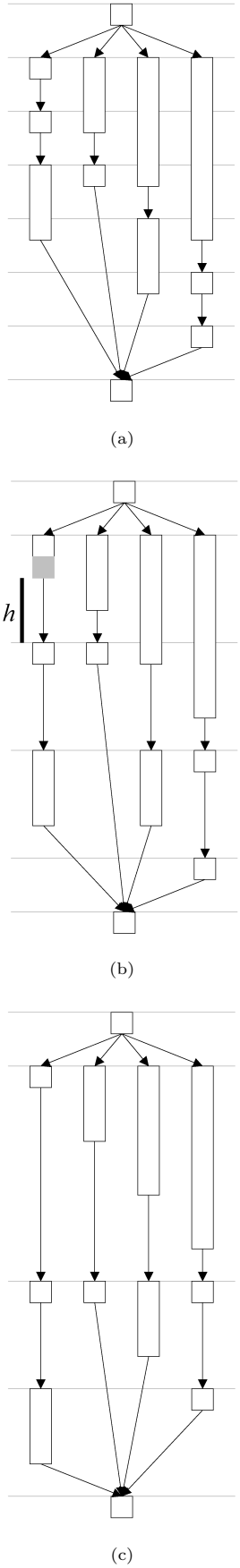


Figure 11: Different values of l_{height} yield different layerings: (a) *size-true* layering ($l_{height} = 0$), (b) $l_{height} =$ "height of line h" (the minimum distance m_d is shown in grey), (c) *layer-maximal* layering. These drawings also show the decrease of the number of layers with increasing value of l_{height}.