Upward Planarization Layout

Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Hoi-Ming Wong*

Chair for Algorithm Engineering, TU Dortmund, Germany

Abstract. Recently, we presented a new practical method for upward crossing minimization [6], which clearly outperformed existing approaches for drawing hierarchical graphs in that respect. The outcome of this method is an *upward planar representation (UPR)*, a planarly embedded graph in which crossings are represented by dummy vertices. However, straight-forward approaches for drawing such UPRs lead to quite unsatisfactory results. In this paper, we present a new algorithm for drawing UPRs that greatly improves the layout quality, leading to good hierarchal drawings with few crossings. We analyze its performance on well-known benchmark graphs and compare it with alternative approaches.

1 Introduction

The visualization of hierarchical graphs representing some natural flow of information is one of the key topics in graph drawing. It has numerous practical applications and received a lot of scientific attention since the very beginning of graph drawing. Formally, we are given a directed acyclic graph (DAG) G and we want to find an *upward drawing* of G, i.e., a drawing of G in which all arcs are drawn as curves monotonically increasing in the vertical direction.

In 1981, Sugiyama et al. [18] proposed their well-known three-phase framework for creating such drawings, which is still widely used:

1. **Layer assignment:** Assign nodes to layers such that arcs point from lower to higher layers; split long arcs spanning several layers by creating *dummy nodes*.
2. **Node Ordering/Crossing reduction:** Order nodes on the layer to reduce the number of arc crossings.
3. **Coordinate assignment:** Assign coordinates to original nodes and dummy nodes (bend points) such that we get only few bend points and short arcs.

A vast amount of modifications and alternatives for the individual steps have been proposed, e.g., Gansner et al. [11] give a LP-based formulations for layer and coordinate assignment. The layer assignment computes a layering which minimize the sum of the vertical edge lengths (i.e., the number of layers an edge spans). The coordinate assignment minimizes the objective function $\sum_{e=(v,w)\in A} w(e) \cdot |X(v) - X(w)|$ where $w(e)$ gives the priority for drawing e vertically and A is the arc set after splitting long arcs. Brandes and Köpf [3] propose an approach which is simpler and faster than [11], but

* Hoi-Ming Wong was supported by the German Research Foundation (DFG), priority project (SPP) 1307 "Algorithm Engineering", subproject "Planarization Practices in Automatic Graph Drawing".

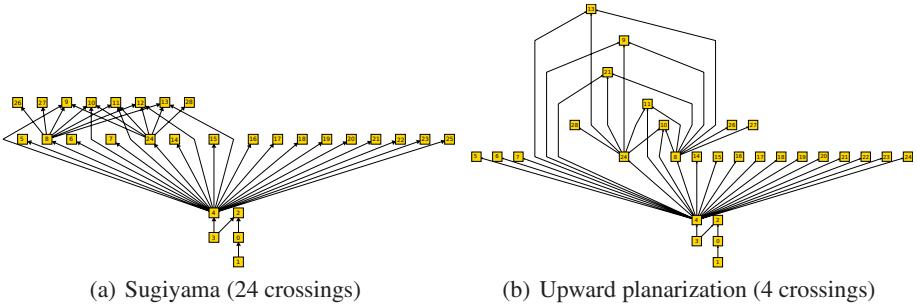
D. Eppstein and E.R. Gansner (Eds.): GD 2009, LNCS 5849, pp. 94–106, 2010.

nevertheless it compute coordinate assignment with similar quality. Brankes et al. [4] investigate the computation complexity of the *width-restricted graph layering problem*. They proved that width-restricted graph layering is NP-hard when taking the dummy nodes into account. Healy and Nikolov give an experimental analysis of existing layering algorithms for DAGs. They also give an ILP formulation which computes a layering with minimum number of dummy nodes with a given upper bound on the width and height of the layering and an branch and cut algorithm to solve it [14,13].

However, a major drawback of Sugiyama's framework could not be solved by any of these modifications: Since layer assignment and crossing reduction are realized as independent steps, the resulting drawing might have many unnecessary crossings caused by an unfortunate layer assignment. A main challenge is to perform crossing reduction *independent* of a layer assignment. First steps to adapt the planarization approach for undirected graphs [1,12] have been presented in [2,7]; Eiglsperger et al. [10] presented the more advanced *mixed upward planarization* approach. However, even the latter approach still needs some kind of layering. Experimental results suggested that this approach produces considerably less crossings than Sugiyama's algorithm. In [6], we presented a novel approach for upward planarization that does not require any layering. We could experimentally show that this new approach clearly outperforms Eiglsperger's mixed upward planarization and Sugiyama's algorithm with respect to crossing reduction.

The output of an upward planarization procedure is an *upward planar representation*, i.e., a representation of the original digraph, in which crossings are replaced by dummy vertices (*crossing dummies*), with a planar embedding and designated external face. In our case, the upward planar representation will always be a single-source digraph; if the input digraph contains multiple sources we introduce a super-source $\hat{s}$ connected to all sources and do not count crossings with arcs incident to $\hat{s}$.

A simple method to draw a DAG by applying upward planarization consists of using Sugiyama's coordinate assignment phase for drawing the upward planar representation, where we use a straight-forwardly obtained layering and the ordering of the nodes on each layer implied by the upward-planar representation and embedding. However, this method produces quite unsatisfactory drawings with too many layers and much too long arcs. The main objective of this paper is to significantly improve on this simple method, by enhancing the computation of layers and node orderings, taking into account the



(a) Sugiyama (24 crossings) (b) Upward planarization (4 crossings)

Fig. 1. Instance *g.29.16* (North DAGs) with 29 nodes and 38 arcs

special roles of crossing dummies. This will allow us to reduce the heights of the drawings and lengths of the arcs substantially, resulting in much more pleasant drawings.

Since upward planarization yields an upward planar representation, we can alternatively also use drawing methods for upward planar digraphs to draw it, cf. [7]. We consider two such algorithms in our experimental study:

Dominance drawings: The linear-time algorithm by Di Battista et al. [9] draws planar st-digraphs with small area on a grid. We apply this algorithm by augmenting the upward planar representation to a planar st-digraph and omitting the augmenting arcs in the final drawing.

Visibility representations: We use the algorithm by Rosenstiehl and Tarjan [17] for computing a visibility representation on the grid. This algorithm is based on bipolar orientations implied by an st-numbering. By augmenting the upward planar representation to an st-planar digraph, we obtain a bipolar orientation such that the resulting drawing is upward planar. Again, we omit augmenting edges in the final drawing.

Fig. 1 shows a relatively small digraph, where the benefits of the new upward planarization approach can be easily seen: While the classical Sugiyama's approach leads to few layers, our approach can expand the layout of the subgraph that looks very congested otherwise.

Upward Planarization. We briefly sketch the planarization approach proposed in [6]. It can be divided into two phases: the feasible subgraph computation and the reinsertion phase. In the first phase the input DAG G is transformed into a single source digraph G' by adding an artificial super source $\hat{s}$ and connecting it to the sources of G. Then we compute a spanning tree T of G' and iteratively try to insert the remaining arcs into T. Thereby, we perform a subgraph feasibility test after each inserted arc e: we do not only test upward planarity but also if all remaining edges can potentially still be inserted (with crossings) in an upward fashion. If the resulting digraph is not feasible in this sense, we add e to a set of deleted arcs B instead. By these operations, we obtain an embedded feasible upward planar subgraph U.

In the second phase, the arcs of B are reinserted into U one after another such that few crossings arise. Thereby, the crossings caused by the reinsertion are replaced by crossing dummies. As a result, we obtain an upward planar representation R of G' (cf. Fig. 2). Note that R is an embedded upward planar single-source digraph. It can easily be augmented to a single-source, single-sink digraph by adding additional arcs (*face-arcs*) and an additional super sink $\hat{t}$ that is connected with the former sinks on the external face.

In the following, R will always be an embedded single-source, single-sink upward planar representation of G. Let v and e be a node and an arc in G, respectively. We denote the corresponding node and arc in R by v_R and e_R, respectively.

2 Upward Planarization Layout Algorithm

The crucial starting point of our algorithm can be stated as follows:

Proposition 1. Given an upward planar representation R of a DAG G, there exists a layering of the nodes of G, a node order per layer, and a node placement including bend points, such that the thereby induced drawing of G *realizes* R, i.e., the crossings arising in G are the ones modeled by R.

We observe that a realizing drawing of G hence follows Sugiyama's framework, but the individual steps do not simply optimize their respective objectives, but follow the overall goal of simulating R. Our algorithm hence divides naturally into the three steps known from Sugiyama's framework, whereby the first two steps are closely related.

As sketched in Sect. 1 (and investigated in the experimental comparison, Sect. 3.1), it is easy to find *some* solution that realizes R. Yet, even if G is only of moderate size, R can become much larger due to the crossing dummies and long arcs. This causes weak runtime performance, many layers, and overall unsatisfactory drawings. Hence our algorithm aims at minimizing the required layers, thereby also reducing the necessary dummy nodes from splitting long arcs.

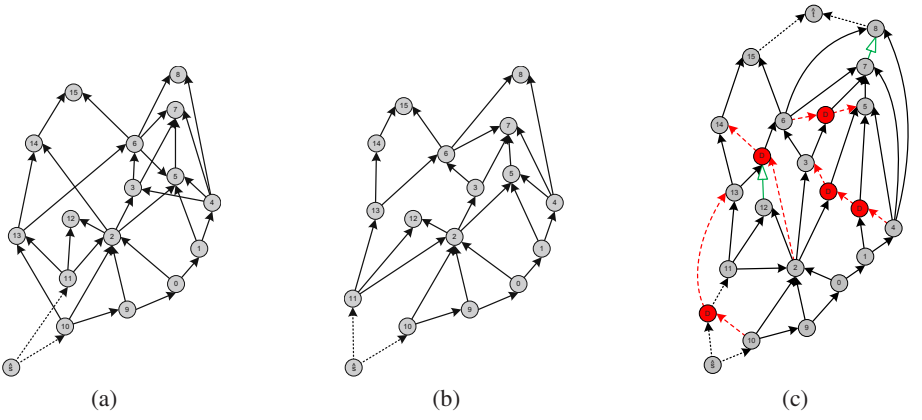
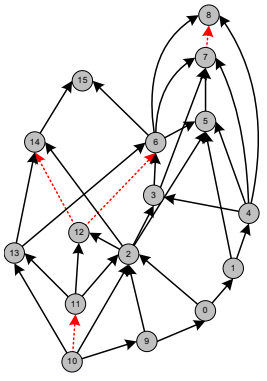


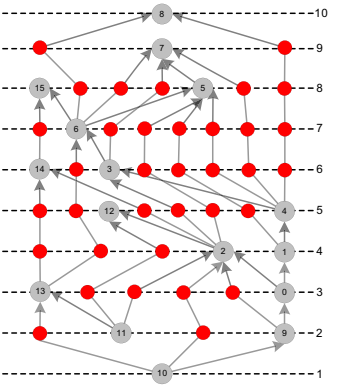
Fig. 2. Illustration of the upward planarization approach by Chimani et al. [6]. (a) input DAG G augmented to G' via the artificial super source $\hat{s}$; (b) embedded feasible subgraph U of G' obtained by deleting the arcs $(10, 13), (2, 14), (4, 3), (6, 5)$; (c) upward planar representation R of G after reinserting the deleted arcs (dashed line). R contains five crossing dummies. By adding face-arcs (drawn with hollow arrow heads) and the super sink $\hat{t}$, R becomes a single-source, single-sink digraph.

2.1 Layer Assignment and Node Ordering

Let H be a copy of G that we will use to obtain a valid layering for G, cf. Fig. 3. For any two nodes $u, v \in V(G)$, we add an auxiliary arc (u, v) to H if: (a) there exists no directed path from u to v in G, but (b) there exists a directed path from u_R to v_R in R. Part (a) prohibits the unnecessary generations of transitive arcs. Part (b), in conjunction with the face-arcs and the single-source, single-sink property of R, ensures that the hierarchical order of R is mapped to H. Since G and R are DAGs, H is also acyclic, and we can use any existing layering algorithm on H. Let $\mathcal{L} = \langle L_1, L_2, ..., L_\ell \rangle$ be the



(a) The auxiliary digraph H with respect to R from Fig. 2. Arcs in H but not in G are drawn as dotted lines.



(b) Layering of H. Long-arc dummies are drawn as smaller circles.

Fig. 3. Layer Assignment and Node Ordering

layering of H, and therefore also for G; i.e., $\dot{\bigcup}_{1\leq i\leq \ell} L_i = V(G)$. We can extend this layering naturally, by splitting any arc that spans more than one layer into a chain of arc segments by introducing dummy nodes (*long-arc dummies*).

Considering the layering $\mathcal{L}$, we now have to arrange the nodes on each layer according to the order induced by R. For this purpose, we consider the order of the arcs around each node, as given by R. In particular, we can recognize the *left incoming* arc for any node v, which is the embedding-wise left-most arc with target v. Note that this arc is defined for each node except for the super source.

Now consider any two disjoint nodes u and v on the same layer. We can decide their correct order using the following strategy: we construct a *left path* p_u from $\hat{s}$ to u_R from back to front, i.e., starting at u_R we select its left incoming arc e as the end of p_u and proceed from the source node of e, choosing its left incoming arc as the second to last arc in p_u, and so on. The construction of p_u ends when we reach the super source, which will always happen as R is a single-sink DAG. Analogously, let p_v be the left path from $\hat{s}$ to v_R.

The paths p_u and p_v may share a common subpath starting at $\hat{s}$; let c_R be the last common node of p_u and p_v, and let e_u and e_v be the first disjoint path elements, respectively. We determine the ordering of u and v directly by the order of e_u and e_v at c_R. For example in Fig. 3, if u_R and v_R are the nodes '4' and '12' respectively (layer 5), then node '0' is the last common node of their left paths, and hence v_R is left of u_R.

Algorithmically, we can consider each layer independently. Introducing an auxiliary digraph A, the above relationship between two nodes on the same layer can be modeled as an arc between these two nodes in A. We can construct correct ordering for the layer, by computing the topological order in A. Note that therefore we do not have to compute the arc direction for all node pairs, but only for the ones that are not already "solved" by other arcs through transitivity.

The above approach already gives a good layering and ordering realizing R. Yet, in order to further improve the solution we introduce two postprocessing strategies:

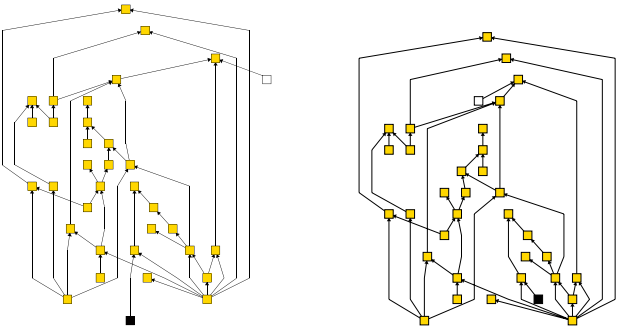


Fig. 4. A drawing of graph *grafo2379.35* (Rome graphs): (left) without postprocessing, (right) after applying source repositioning (white node) and long-arc dummy reduction (black node)

Long-Arc Dummy Reduction. A *dominated subgraph* of G w.r.t. a node s is the subgraph induced by the nodes v for which G contains a directed path from s to v. Most layering algorithms—in particular also the optimal LP-based approach [11]—will put the nodes on the lowest possible layer. While this is generally a good idea, this approach can be counter-productive in the context of the super source node that will be removed from the final drawing: Since every source node s in G is attached to the super source node $\hat{s}$ (which is on the lowest layer), s may end up very low in the drawing, even though most of its dominated subgraph requires higher layers, hence introducing long arcs.

We tackle this problem using an approach similarly to the *promotion node method* by Nikolov and Tarassov [15] by re-layering parts of dominated subgraphs after the removal of $\hat{s}$ incrementally, without modifying the hierarchical order induced by R. Layers that become empty by these operations can be removed afterwards:

1. **For Each** source s in G (in decreasing order of their layer index j):
   (a) Mark the subgraph dominated by s. Let M_i be the marked nodes on layer L_i $(1 \leq i \leq \ell)$.
   (b) **For** $i = j + 1$ **To** ℓ:
   If M_i are all long-arc dummies **Then**
   1. Remove the nodes M_i and lift the marked subgraph on the layers below L_i by one layer.
   2. **If** the new layering causes more edge crossings **Or** more long-arc dummies **Then Undo** step 1. and **Return**

Repositioning the Sources. Since our upward planarization algorithm considers G augmented with $\hat{s}$ and inserts additional arcs considering a fixed embedding, the final upward planar representation may contain artifacts in form of seemingly unnecessary crossings, see, e.g., node '5' in Fig. 4. To overcome this, we sift each source s through all possible positions on its layer and choose the position where it causes the fewest crossings.

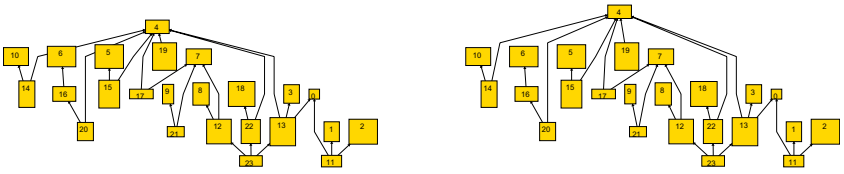


Fig. 5. A drawing of graph *grafo159.24* (Rome graphs) with random node sizes: without (left) and with (right) our bending arcs method and individual layer distance assignment

2.2 Coordinate Assignment

After the previous steps we get a correct layering and node ordering, realizing R. Conceptually, we can use *any* coordinate assignment strategy (e.g., [11,5]) known for Sugiyama's layout; it will always maintain the given number of crossings. Such strategies assign horizontal coordinates to the nodes, while maintaining the given ordering. The aim is to generate drawings such that the subdivided long arcs are drawn as vertical straight lines for their most part.

Yet, when considering the hard-to-measure "beauty" or "readability" of the resulting drawings, we realize that we can improve on traditional coordinate assignment strategies as they usually do not accommodate for the following two drawing problems:

- *node-arc crossings*: A line segment connecting nodes or bend-points between layer L_i and L_{i+1} may cross through some nodes of these two layers. This can easily happen when node sizes are relatively large compared to the layer distance.
- *long-line segments*: The general direction of upward drawings should naturally be along the vertical direction. Yet, there can be arc segments between some layers L_i and L_{i+1} which are very long since they span a large horizontal distance. Such arcs can make Sugiyama-style drawings hard to read.

Fig. 5 shows the benefit of the two strategies described below. Note that these strategies are not only applicable to our layout algorithm, but to any Sugiyama-style layout.

Vertical Coordinates. Usually, the vertical coordinates for the nodes on layer L_i are simply given by $\delta \cdot i$, where δ is the minimal layer distance. Yet, often we may prefer larger distances between layers in order to improve readability: larger distances counter both above problems, but in our context we are in particular interested in long-line segments—we will discuss how to tackle node-arc crossings in the paragraph hereafter.

Buchheim et al. [5] propose a solution for variable layer distance computing which dependents on the gradient of the line segments. However, our experimental results show that drawing DAGs using upward planarization tends to produce drawings with large height. Therefore we use a different approach which limits the maximal layer distance to 3δ.

Let σ_i be the number of arcs between L_i and L_{i+1} whose horizontal dimension is at least 3δ. Then we set the vertical distance between these two layers to $(1 + \min\{\sigma_i/4, 2\})\delta$.

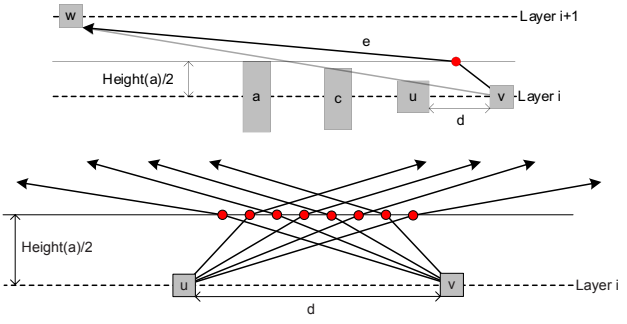


Fig. 6. (top) Avoiding node line-overlapping by introducing new bend-point into an arc e. (bottom) The horizontal coordinates of the bend-points must be all distinct, even if all involved arcs require a bend.

Bending Arcs. While enlarging the layer distance also helps to prevent node-line crossings, the necessary height increase is usually not worth it, from the readability perspective. We therefore propose a strategy that allows trading additional bend-points for layer distance. The strategy can be parameterized to find one's favorite trade-off between these two measures.

Let $X(v)$ and $Y(v)$ denote the horizontal and vertical coordinates of a node v, respectively. An arc (or line segment) $e = (v, w)$ is pointing upward from left to right (right to left) if $X(v) < X(w)$ ($X(v) > X(w)$, resp.). Since purely vertical line segments cannot cross through nodes, we distinguish four cases: e is pointing upward from right to left (or left to right) and v (or w) is a node on layer i. In all these cases, e has to bend if it overlaps some nodes of L_i. However, bending e might cause additional crossings. To avoid this, we also have to bend the line segments that cross the just bended line. W.l.o.g., we only discuss the case $X(v) > X(w)$ with $v \in L_i$. The other cases can be solved analogously.

Let $\mathrm{width}(v)$ and $\mathrm{height}(v)$ denote the width and height of the bounding box of a node v. Let a be the node on layer L_i with the highest bounding box, and let $\alpha := \mathrm{height}(a)/2$. If v is a bend-point, we do not need to introduce an additional bend. Instead we move v upwards by α. It can happen that v was already shifted downwards before, due to one of our other cases. Then, we bend e by introducing a new bend-point b and set $X(b) := X(v)$ and $Y(b) := Y(v) + 2\alpha$.

Assume v is not a bend-point. Then we have to introduce a bend point along e. Thereby we have to consider that other arcs might also get rerouted, and accommodate enough space for them as well such that no two bend points may coincide. In particular, it might be that the arcs leaving v's left neighbor to the right might also require additional bend points (cf. Fig. 6). Let u be the left neighbor of v on L_i and $d := X(v) - X(u) - \mathrm{width}(v)/2 - \mathrm{width}(u)/2$ their inner distance. Let r be the number of line segments adjacent to v and pointing from right to left; among these, assume that e is the j-th segment when counting from left to right. Let q be the number of line segments adjacent to u and pointing from left to right. Then, $\Delta := \left(\frac{d}{q+r+1}\right)$ gives the distances between the potential bend points and the coordinates of the new bend-point b are given by $X(b) := X(u) + \frac{\mathrm{width}(u)}{2} + \Delta \cdot (j + \min\{q, j-1\})$ and $Y(b) := Y(v) + \alpha$.

3 Experiments

We investigate the quality of our new algorithm in comparison with known algorithms. We first compare different approaches to draw a computed upward planar representation, i.e., if the crossing number is the most important factor in our drawing. Afterwards, we also compare our approach to Sugiyama's traditional framework.

All algorithms are implemented in the free and open-source (GPL) *Open Graph Drawing Framework (OGDF)* [16]. The experiments were conducted on an IBM Thinkpad with an Intel Pentium M 1.7Ghz and 1GB RAM. We use the following well-known benchmark sets:

Rome Graphs: The Rome graphs [8] are a widely used benchmark set in graph drawing, obtained from a basic set of 112 real-world graphs. The benchmark contains 11528 instances with 10–100 nodes and 9–158 edges. Although the graphs are originally undirected, they have been used as directed graphs by artificially directing the edges according to the node order given in the input files, see, e.g., [10,6].

North DAGs: The North DAGs[1] have been introduced in an experimental comparison of algorithms for drawing DAGs [7]. The set consists of 1277 DAGs collected by Stephen North. The digraphs are grouped into 9 sets, where the first set contains digraphs with 10 to 20 nodes and the i-th set contains $10i + 1$ to $10(i + 1)$ nodes for $i = 2, \ldots, 9$.

3.1 Planarization Layouts

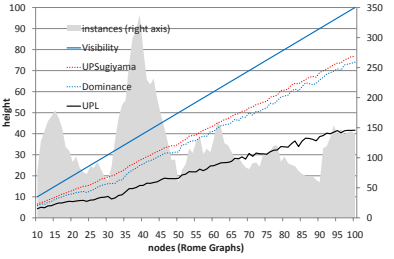
As outlined in the introduction, there are various other possibilities to draw an upward planar representation R of a digraph. Therefore, we use R as the input for the following algorithms. After computing the drawing, we can replace the dummy nodes by usual arc crossings and remove the face-arcs and the super source/sink. By this approach we guarantee that the resulting drawing realizes the specified representation.

We denote our new algorithm by Upward Planarization Layout (*UPL*). We compare it to the *Dominance* drawing style [9], the *Visibility* Representation drawing style [17], and to a straight-forward application of Sugiyama's framework (*UPSugiyama*). For the latter, we use Optimal Ranking [11] for layering, extract the node orders directly from the upward planar representation, and use Fast Hierarchy [5] for coordinate assignment.

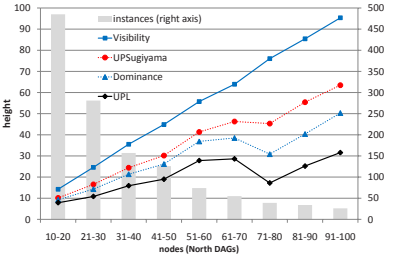
Fig. 7(a)–(d) give the height and width of the resulting drawings, respectively, averaged over the digraphs with the same number of nodes. For a fair comparison between the approaches, disregarding any differences due to spacing parameters, we have: The *height* of a drawing is the number of required layers, in case of *UPL* and *UPSugiyama*, and the number of vertical grid coordinates in case of *Dominance* and *Visibility*. The *width* of a drawing is the maximum number of elements per layer, or per horizontal grid line, respectively.

For a fair runtime comparison (Fig. 7(e) and (f), time in seconds), we use the same coordinate assignment algorithm for *UPL* as for *UPSugiyama*. This choice is due to the fact that the alternative ILP approach [11] would require too much time for *UPSugiyama*, which has to consider very large digraphs due to the crossing and long-arc

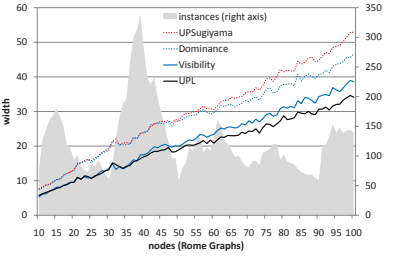
[1] www.graphdrawing.org/data/index.html



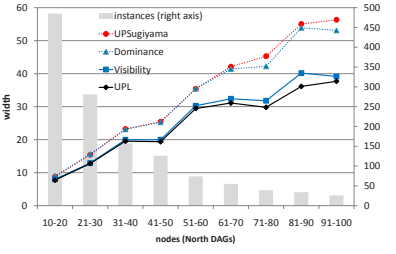
(a) Height (Rome graphs)



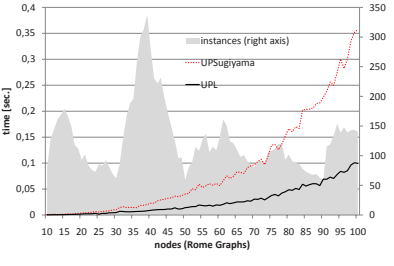
(b) Height (North DAGs)



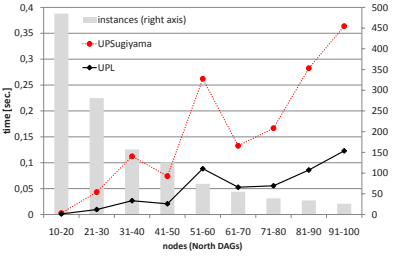
(c) Width (Rome graphs)



(d) Width (North DAGs)



(e) Time (Rome graphs)

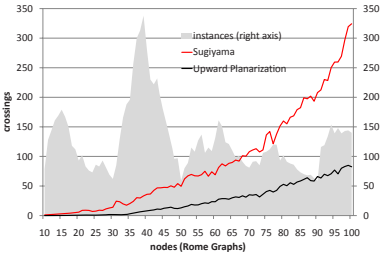


(f) Time (North DAGs)

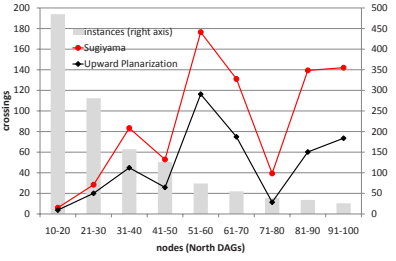
Fig. 7. Comparison with other algorithms to draw upward planar representations; the data points represents average values for the corresponding node groups

dummies. Note that the above height and width measures are invariant under the choice of coordinate assignment. We omit the runtime figures for the linear-time algorithms *Dominance* and *Visibility*, as they are usually below any measurable threshold.

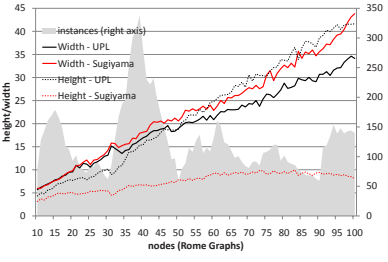
We see that our new approach clearly outperforms all other approaches on the geometry measures, independent of the benchmark set. Also, the runtime comparison shows *UPL* as the clear winner compared to *UPSugiyama*.



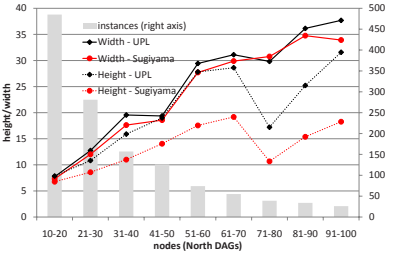
(a) Number of crossings (Rome graphs)



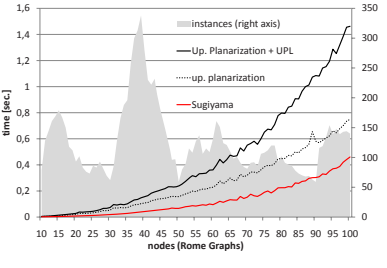
(b) Number of crossings (North DAGs)



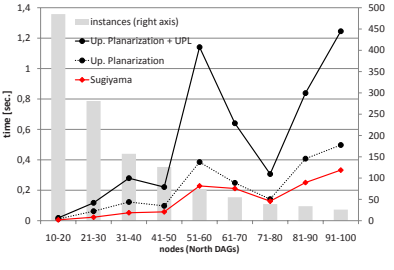
(c) Height and width (Rome graphs)



(d) Height and width (North DAGs)



(e) Time (Rome graphs)



(f) Time (North DAGs)

Fig. 8. Comparison with Sugiyama's algorithm; the data points represents average values for the corresponding node groups

3.2 Comparison with Traditional Sugiyama

Finally, we can investigate how much the requirement of having a drawing with few crossings costs in terms of other quality measures. Therefore we compare *UPL* to a traditional Sugiyama approach that is not bound to a specific upward planar representation. For the latter we use the experimentally most competitive combination by layering via Optimal Ranking [11], using the barycenter heuristic for the crossing

reduction step (best of 5 randomized runs), and assigning the coordinates via the exact LP approach [11]. For a fair comparison, *UPL* also uses the latter coordinate assignment algorithm. This time, the runtime of *UPL* furthermore includes the computation of the planarization, as this step is not necessary for Sugiyama. Note that the implementation of the planarization was vastly improved compared to the performance given in [6].

Clearly, the number of crossings in the pure Sugiyama approach are much higher (Fig. 8(a) and (b)), which is consistent with the findings in [6]. As shown in Fig. 8(c) and (d), our *UPL* drawings are of course higher than Sugiyama's, by construction. Yet we observe that the difference is not as large as one might have expected, and *UPL* seems to be a good fit also with respect to this measure, if a small number of crossings is an important issue. We can also see that *UPL* has certain advantages over Sugiyama's approach: A strong packing into few layers will usually require a wider drawing than our planarization. Furthermore, such few layers can in fact be counterproductive from the point of readability, see, e.g., Fig. 1. Overall, we can observe that *UPL* obtains a more balanced aspect ratio than Sugiyama's approach.

In terms of running time (Fig. 8(e),(f)), we see that while Sugiyama's approach is generally faster, *UPL* is not too slow either, requiring below 1.5 seconds for the large instances.

4 Conclusion

Traditional methods of drawing DAGs consider the number of crossings only as a second order priority. If it is of highest priority, one has to use algorithms to draw upward planar representations. Our algorithm constitutes the first such algorithm that takes the special crossing nodes into account. As our experiments show, it generates drawings that are preferable over the other known methods to draw a given upward planar representation. Furthermore, the drawing is also comparable to Sugiyama's approach with respect to other quality measures, while offering a smaller number of crossings.

References

1. Batini, C., Talamo, M., Tamassia, R.: Computer aided layout of entity relationship diagrams. J. Syst. Software 4, 163–173 (1984)
2. Di Battista, G., Pietrosanti, E., Tamassia, R., Tollis, I.G.: Automatic layout of PERT diagrams with X-PERT. In: Proc. IEEE Workshop on Visual Languages, pp. 171–176 (1989)
3. Brandes, U., Köpf, B.: Fast and simple horizontal coordinate assignment. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 31–44. Springer, Heidelberg (2002)
4. Branke, J., Leppert, S., Middendorf, M., Eades, P.: Width-restricted layering of acyclic digraphs with consideration of dummy nodes. Inf. Process. Lett. 81(2), 59–63 (2002)
5. Buchheim, C., Jünger, M., Leipert, S.: A fast layout algorithm for k-level graphs. In: Marks, J. (ed.) GD 2000. LNCS, vol. 1984, pp. 229–240. Springer, Heidelberg (2001)
6. Chimani, M., Gutwenger, C., Mutzel, P., Wong, H.-M.: Layer-free upward crossing minimization. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 55–68. Springer, Heidelberg (2008)

7. Di Battista, G., Garg, A., Liotta, G., Parise, A., Tamassia, R., Tassinari, E., Vargiu, F., Vismara, L.: Drawing directed acyclic graphs: An experimental study. Int. J. Comput. Geom. Appl. 10(6), 623–648 (2000)
8. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. Comput. Geom. Theory Appl. 7(5-6), 303–325 (1997)
9. Di Battista, G., Tamassia, R., Tollis, I.G.: Area requirement and symmetry display of planar upward drawings. Discrete Comput. Geom. 7(4), 381–401 (1992)
10. Eiglsperger, M., Kaufmann, M., Eppinger, F.: An approach for mixed upward planarization. J. Graph Algorithms Appl. 7(2), 203–220 (2003)
11. Gansner, E., Koutsofios, E., North, S., Vo, K.-P.: A technique for drawing directed graphs. Software Pract. Exper. 19(3), 214–229 (1993)
12. Gutwenger, C., Mutzel, P.: An experimental study of crossing minimization heuristics. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 13–24. Springer, Heidelberg (2004)
13. Healy, P., Nikolov, N.S.: A branch-and-cut approach to the directed acyclic graph layering problem. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 98–109. Springer, Heidelberg (2002)
14. Healy, P., Nikolov, N.S.: How to layer a directed acyclic graph. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 16–30. Springer, Heidelberg (2002)
15. Nikolov, N.S., Tarassov, A.: Graph layering by promotion of nodes. Discrete Applied Mathematics 154(5), 848–860 (2006)
16. OGDF – the Open Graph Drawing Framework. Technical University of Dortmund, Chair of Algorithm Engineering, `http://www.ogdf.net`
17. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete Comput. Geom. 1(1), 343–353 (1986)
18. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Trans. Sys. Man. Cyb. 11(2), 109–125 (1981)