

Layer-Free Upward Crossing Minimization

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

Department of Computer Science, Technical University of Dortmund, Germany
{markus.chimani, carsten.gutwenger, petra.mutzel,
hoi-ming.wong}@cs.uni-dortmund.de

Abstract. An upward drawing of a DAG G is a drawing of G in which all edges are drawn as curves increasing monotonically in the vertical direction. In this paper, we present a new approach for upward crossing minimization, i.e., finding an upward drawing of a DAG G with as few crossings as possible. Our algorithm is based on a two-stage upward planarization approach, which computes a feasible upward planar subgraph in the first step, and re-inserts the remaining edges by computing constraint-feasible upward insertion paths. An experimental study shows that the new algorithm leads to much better results than existing algorithms for upward crossing minimization, including the classical Sugiyama approach.

1 Introduction

Drawing hierarchical graphs is one of the fundamental issues in graph drawing, having received a lot of attention in the past. Given a directed acyclic graph (DAG) G , we are interested in an *upward drawing* of G , i.e., a drawing of G in which all edges are drawn as curves, monotonically increasing in the vertical direction. This problem has numerous applications and occurs whenever a natural flow of information shall be visualized.

Surprisingly, the state-of-the-art in drawing DAGs still facilitates the general framework by Sugiyama et al. in 1981 [12], which consists of three steps:

1. **Layer Assignment:** Assign the nodes to layers such that edges point from lower to higher layers. Split long edges spanning several layers such that edges connect only nodes on neighboring layers.
2. **Crossing Reduction:** Reorder the nodes on each layer with the objective to reduce the number of edge crossings.
3. **Coordinate Assignment:** Assign final node (and bend-point) coordinates.

The individual steps are usually solved heuristically. Several refinements and improvements to this approach have been published; most notable are the work by Gansner et al. [8] and the fast Sugiyama implementation by Eiglsperger et al. [7]. Nevertheless, one inherent drawback of the framework is not overcome by any of these modifications: assigning nodes to fixed layers in the first step can severely affect the subsequent crossing minimization step, requiring edge

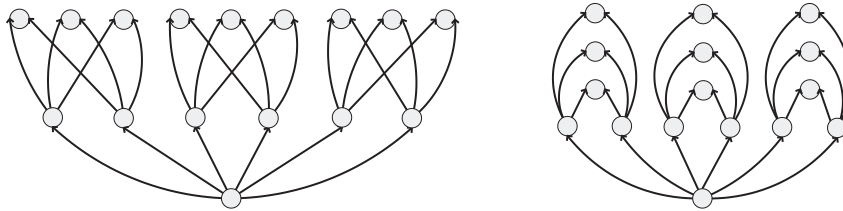


Fig. 1. An unfortunate layering (left) can force unnecessary crossings. In this example the graph is even upward planar (right).

crossings that would be unnecessary if a “better” layer assignment had been chosen, cf. Figure 1.

In this paper, we propose a replacement for the first two steps of the Sugiyama framework, thus combining layer assignment and crossing minimization in order to obtain drawings with fewer crossings. We borrow ideas from the *planarization approach* [1, 10], which is the most successful heuristic for minimizing crossings in undirected graphs. This approach computes a large planar subgraph in the first step, and then tries to re-insert the remaining edges one-by-one, each time replacing the required crossings with dummy vertices, so that the graph remains planar. The sequence of edges crossed while inserting an edge is also called an *insertion path*. The final outcome is a *planarized representation* of the input graph in which edge crossings are represented as dummy vertices of degree four.

However, adapting this approach to upward drawings is by far not straightforward. First of all—while planarity can be tested efficiently—testing upward planarity is NP-complete [9]; on the other hand, upward planarity can efficiently be tested for graphs with a single source [2], so-called *sT-graphs*. Secondly, while any planar subgraph is suitable as a starting point in the undirected case, further constraints are necessary for upward drawings. And finally, it is not sufficient to find an upward insertion path for an edge independent of the remaining edges. Details and examples on these challenges are given in Section 2.

First successful results on applying the idea of upward planarization are given by Eiglsperger et al. [6] in the context of mixed-upward graphs (graphs in which only a subset of the edges needs to be drawn upward). However, their proposed heuristic for upward edge insertion still needs to layer the graph. Our aim is to completely forgo layering the graph in the crossing minimization step.

The paper is organized as follows. In Sections 2–4, we describe our new approach to upward planarization, where Section 2 gives an informal overview on the algorithm. Section 3 provides necessary formal definitions and describes our basic tool for modeling the dependencies between the edges to be inserted. Section 4 details the upward edge insertion procedure. Finally, an experimental study in Section 6 proves the superiority of the new approach with respect to solution quality compared to existing algorithms.

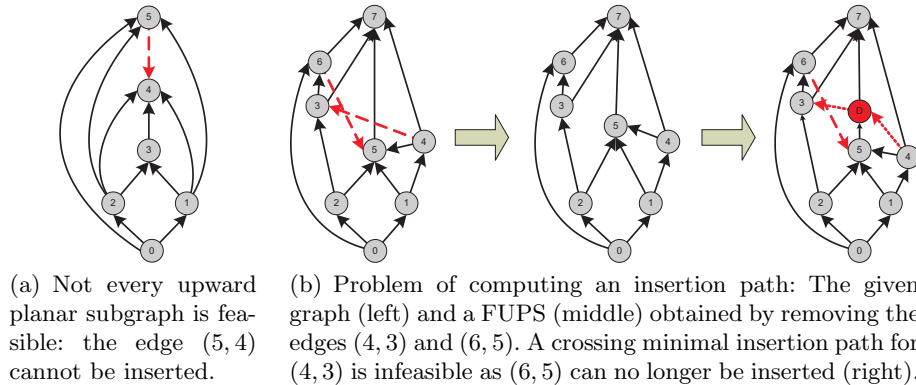


Fig. 2. Problems with simple edge insertion approaches.

2 Upward Planarization Approach

In the following, we are given an sT -graph $G = (V, A)$ that we want to planarize with as few crossings as possible. If G would have multiple sources, we can add a super source node \hat{s} , connect it to all original sources, and set the costs for crossing these additional edges to 0. Like the traditional planarization approach for undirected graphs, our algorithm consists of two stages: in the first we will construct a *feasible* upward planar subgraph $U = (V, A')$, formally defined in Section 3. In the second step, we will iteratively insert the edges not yet in U so that few crossings arise; these crossings are replaced by dummy nodes so that the graph in which edges are inserted can always be considered planar. Inserting an edge e into a planar graph thereby means that all arising crossings will lie on e ; we do not introduce additional crossings purely on the planar graph itself.

The main challenge with this approach is that—in contrast to the approach for undirected graphs—the edge insertion steps are not independent of each other. In particular:

- Assume we construct the upward planar subgraph straight-forwardly by adding edges to an initially empty subgraph; after each edge we test for upward planarity. The process stops when no more edges can be inserted without losing upward planarity. As shown in Figure 2(a), we may not be able to insert the remaining edges *at all*, no matter how many crossings we may use and which upward embedding we choose for the subgraph. Hence we need to identify a *feasible* upward planar subgraph (FUPS).
- Similarly, even if we have a FUPS, we cannot easily insert edges iteratively into it in a crossing minimal fashion, without taking the not-yet inserted edges into account. Figure 2(b) shows that, even though it is possible to insert both edges into U , inserting one edge inconsiderately may make inserting the other one impossible.

Algorithm 1 Upward planarization algorithm

Require: sT -Graph $G = (V, A)$ **Ensure:** Upward-planar planarized representation of $G = (V, A)$

- 1: Identify spanning tree $U = (V, A')$ of G , directed from s outwards
 - 2: \triangleright Compute feasible upward planar subgraph (FUPS)
 - 3: $B := \emptyset$
 - 4: **for** each $e \in A \setminus A'$ **do**
 - 5: $U' := U + e$
 - 6: **if** \exists upward planar embedding Γ' of U' **then**
 - 7: **if** $\mathcal{M}(\Gamma')$ is acyclic **then**
 - 8: $U := U', \Gamma := \Gamma'$
 - 9: **continue**
 - 10: $B := B \cup \{e\}$
 - 11: \triangleright Non-planar edge insertion
 - 12: **for** each $e \in B$ **do**
 - 13: Compute insertion path p for e into Γ that will ensure property **(M)**
 - 14: Insert e along p , replacing crossings by dummy nodes \rightarrow new U, Γ .
 - 15: **Property (M):** $\mathcal{M}(\Gamma)$ is acyclic
-

We resolve these problems by introducing a *merge graph* \mathcal{M} , i.e., a graph representing not only the current (probably planarized) subgraph U , but also modeling all edges that are yet to be inserted. The special properties of \mathcal{M} , cf. Section 3, allow us to use a simple acyclicity test on it to determine whether all remaining edges will be insertable. Algorithm 1 gives an overview on our upward planarization algorithm. The next section will investigate the merge graph, its applicability and its computation further; Section 4 will center on the crossing-minimal edge insertion (lines 15–19 in Algorithm 1).

3 Feasible Graphs and Paths

Let $G = (V, A)$ be an sT -graph and $U = (V, A')$ an upward planar subgraph of G . We define:

Definition 1 (Upward Insertion Path). *An insertion path p_1 w.r.t. some edge $e_1 = (x_1, y_1) \in A \setminus A'$ is an ordered list of edges $a_1, \dots, a_\kappa \in U$ such that the graph U_1 obtained from realizing p is upward planar. The realization works as follows: we split the edges a_1, \dots, a_κ obtaining the dummy nodes d_1, \dots, d_κ , and add the edges $(x_1, d_1), (d_1, d_2), \dots, (d_\kappa, y_1)$ representing e_1 .*

Let Γ_1 be an upward planar embedding of U_1 . We say Γ_1 induces an upward planar embedding Γ of U , which is obtained by reversing the realization procedure while maintaining the embedding.

Definition 2 (Upward Insertion Sequence). *An upward insertion sequence is a sequence of k upward insertion paths for all edges $A \setminus A'$. Thereby the first edge in the sequence is inserted into U —introducing dummy nodes—which results*

in an upward-planar graph U_1 . The second edge is then inserted into U_1 , etc. After realizing all insertion paths, we hence obtain a final upward planar graph U_k , which is a planarized representation of G .

In the following, let Γ be a fixed upward planar embedding of U .

Definition 3 ((Constraint) Feasible Upward Insertion Path). *The upward insertion path p_1 is feasible w.r.t. Γ , if there exists an upward planar embedding Γ_1 of the realizing graph U_1 which induces Γ . It is constraint feasible w.r.t. Γ , if it furthermore allows an upward insertion sequence for the edges $A \setminus \{A' \cup \{e_1\}\}$ into U_1 such that there exists an upward planar embedding Γ_k of the final graph U_k which induces Γ . We say the path p_1 is (constraint) minimal if it requires the fewest crossings over all (constraint) feasible insertion paths.*

Definition 4 (Feasible Upward-Planar Subgraph (FUPS) and Embedding). *An upward planar subgraph $U = (V, A')$ of $G = (V, A)$ is feasible if there exists an insertion sequence. An upward planar embedding Γ of a FUPS U is feasible if there exists an insertion sequence such that Γ_k induces Γ .*

In an upward planar embedding Γ we can define a *sink-switch* [3] w.r.t. some face f , as a node v incident to f for which there exists no edge incident to f and starting at v . Among all sink-switches of an inner face f , we can identify exactly one *top* sink-switch, i.e., the sink-switch which has to be drawn above all other sink-switches. Analogously we can define a *source-switch* as a node v incident to f for which there exists no edge incident to f and ending at v .

Definition 5 (Merge Graph). *The Merge Graph $\mathcal{M}(\Gamma)$ of U with respect to Γ is constructed as follows:*

1. *We start with $\mathcal{M}(\Gamma)$ being a copy of G .*
2. *For each internal face f of Γ , we add an arc from each non-top sink-switch of f to the top sink-switch of f . We call these edges sink arcs.*

Let v_1 and v_2 be two nodes of G . If there exists a non-empty path from v_1 to v_2 , we say v_1 *dominates* v_2 and denote this by $v_1 \rightsquigarrow v_2$. If there does not exist such a path we write $v_1 \not\rightsquigarrow v_2$. Considering some specific upward drawing of G , we denote by $v_1 \prec v_2$ that v_1 is drawn lower than v_2 . An upward planar drawing clearly requires $v_1 \prec v_2$ if $v_1 \rightsquigarrow v_2$. On the other hand, even for any fixed upward planar embedding Γ , there always exists an upward planar drawing with respect to Γ with $v_1 \prec v_2$ if $v_2 \not\rightsquigarrow v_1$.

Lemma 1 (Feasibility Lemma). *The merge graph $\mathcal{M}(\Gamma)$ is acyclic if and only if there exists an insertion sequence such that the resulting graph is upward planar.*

Proof. \exists Seq. $\implies \mathcal{M}(\Gamma)$ acyclic: Consider an upward planar drawing \mathcal{D}_k of U_k with respect to Γ_k . We replace all dummy nodes with crossings and delete the edges $A \setminus A'$ in \mathcal{D}_k . The graph associated with the new drawing \mathcal{D} is the upward

planar subgraph U and the embedding induced by it is Γ . For each face f , we draw an edge from each non-top sink-switch to its according top sink-switch. As Γ is upward planar, these edges are clearly oriented upwards in the drawing. Finally, we reintroduce the edges $A \setminus A'$ into \mathcal{D} , drawing them exactly as in \mathcal{D}_k . By these operations we obtained a drawing of the merge graph, and as all edges in the resulting drawing are oriented upwards, the merge graph is acyclic.

$\mathcal{M}(\Gamma)$ acyclic $\implies \exists$ Seq.: Let \mathcal{N} be the graph $\mathcal{M}(\Gamma)$ without the edges $A \setminus A'$. \mathcal{N} can be embedded like Γ , embedding the sink arcs planarly within their corresponding faces. Let $\Gamma_{\mathcal{N}}$ be the resulting embedding. Let $e_i = (x_i, y_i)$, $1 \leq i \leq k$, be the edges not in U . Since $\mathcal{M}(\Gamma)$ contains these edges and is acyclic we know that $y_i \not\prec x_i$ in \mathcal{N} , for all $1 \leq i \leq k$. Hence, considering any edge e_i individually, we can find a drawing with respect to Γ where $x_i \prec y_i$.

We show that this holds for all edges together by induction over the number of edges to be inserted. We start with any upward drawing of \mathcal{N} which is stepwise modified. Consider the drawing \mathcal{D}_j of \mathcal{N} in step j where we have $x_i \prec y_i$ for all $1 \leq i < j$. If $x_j \prec y_j$ there is nothing to do in this step, so assume $y_j \prec x_j$. Since $\mathcal{M}(\Gamma)$ is acyclic we know that $y_j \not\prec x_j$ and hence there exists a drawing with $x_j \prec y_j$. We show that we can realize the latter condition without violating the respective order for e_1, \dots, e_{j-1} .

Considering $\mathcal{M}(\Gamma)$, let Y_j be the nodes dominated by y_j , i.e., $y_j \rightsquigarrow z$ in $\mathcal{M}(\Gamma)$ for all $z \in Y_j$. Clearly, $x_j \notin Y_j$ as $\mathcal{M}(\Gamma)$ is acyclic. In \mathcal{D}_j , we will move y_j and all nodes Y_j such that y_j is above x_j . Due to the sink arcs, \mathcal{N} does not have any maximal two-connected component C that is within another such component. Hence the upward shift will not result in any crossings.

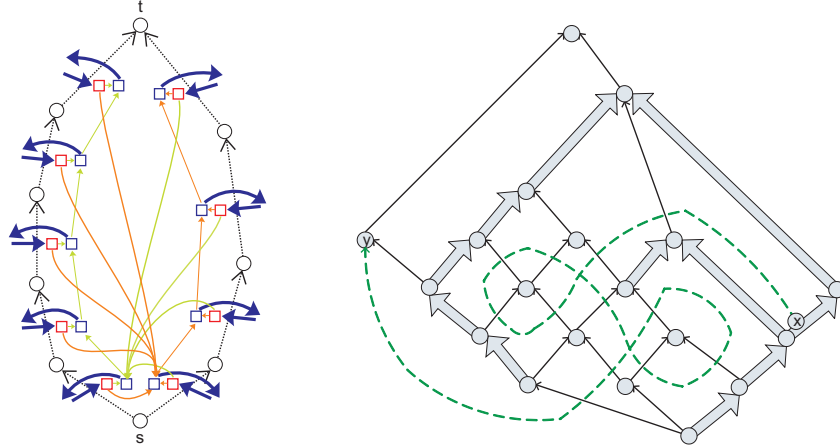
Assume that the shift would invalidate $x_i \prec y_i$ for some i . This would mean that x_i would be in Y_j but y_i would not. But since (x_i, y_i) is an edge in the merge graph this cannot be the case. Therefore the shift in step j ensures an upward planar drawing of \mathcal{N} where all edges e_1, \dots, e_j can be drawn into in an upward fashion using straight lines. We can simply extract insertion paths from the final drawing \mathcal{D}_k to generate a valid insertion sequence. \square

Corollary 1. *An upward planar embedding Γ of a FUPS U is feasible if and only if the corresponding merge graph $\mathcal{M}(\Gamma)$ is acyclic.*

4 Upward Edge Insertion

We now consider the problem of inserting edges into a FUPS with few crossings, by iteratively adding the edges not in the FUPS. In the following we are again given an sT -graph $G = (V, A)$, a FUPS $U = (V, A')$ and an embedding Γ of U . Let $e_i = (x_i, y_i)$, for $1 \leq i \leq k = |A \setminus A'|$ be the edges not in U , and let p_1 be a feasible upward insertion path for e_1 into U with respect to Γ . Realizing p_1 results in a graph U_1 with embedding Γ_1 . Formally we can define the arising problem per edge:

Definition 6 (Constraint Upward Edge Insertion Problem with Fixed Embedding). *Given an sT -graph $G = (V, A)$, a FUPS $U = (V, A')$, a feasible*



(a) Routing sub-network of a single internal face. The dotted lines are the edges of the underlying graph \hat{U} . The bold edges are *crossing arcs*.

(b) Routing in \hat{U} without locking can result in infeasible insertion paths. The thick gray edges denote substructures which are expensive to cross. The dashed line denotes the shortest path in the underlying routing network which makes loops and is thus infeasible as an upward insertion path.

Fig. 3. Finding feasible upward insertion paths using the routing network.

upward embedding Γ of U and an edge $e_1 \in A \setminus A'$. The constraint upward edge insertion problem with fixed embedding is to find a constraint minimal upward insertion path p_1 for e_1 into U with respect to Γ and the edges $A \setminus \{A' \cup \{e_1\}\}$.

4.1 Routing Network

In order to compute an upward insertion path for $e_1 = (x_1, y_1)$ we use a *routing network* R . For the traditional edge insertion problem, i.e., without considering upward planarity, we simply use the bidirected dual graph of U with respect to Γ , augmented with the start and end nodes of e_1 . Due to the required upward planarity we have to use a more heavily augmented routing network.

Firstly, we augment U and Γ with the sink arcs as we did for the merge graph. Furthermore, we add a super sink node \hat{t} and *super sink arcs* (t, \hat{t}) , for each sink on the outface of Γ . We call the resulting graph \hat{U} with its upward planar embedding $\hat{\Gamma}$ (inducing Γ). The augmentation guarantees that all faces in $\hat{\Gamma}$ are *simple*, i.e., they have exactly one source- and one sink-switch.

We do not represent single faces as single nodes in R but as well-structured sub-networks, as shown in Figure 3(a) for an internal face; the sub-network for the external face is analogous. Such a sub-network guarantees that when we enter a face f over some edge, we can only leave f either above that edge or on the other side of f . We call the edges that are the dual of some edge of \hat{U} the *crossing arcs*, as a path over these arcs crosses an edge of \hat{U} .

Finally, we add nodes x^* and y^* to R which will be the start and end node of the insertion path, corresponding to x_1 and y_1 , respectively. Let A_x and A_y be the crossing arcs corresponding to edges starting at x_1 or ending at y_1 , respectively. We add edges from x^* to each target node of the arcs A_x , and edges from each source node of the arcs A_y to y^* . We have:

Lemma 2. *The routing network R has $\mathcal{O}(|V|)$ nodes and edges.*

To use R as a routing network, we assign a cost of 1 to each crossing arc which corresponds to an edge in U , i.e., not to sink arcs in \hat{U} . All other edges in R have cost 0.

4.2 Locking Edges

The routing network by itself is not strong enough to guarantee that the shortest path between x^* and y^* corresponds to a feasible upward insertion path, cf. Figure 3(b): the shortest path may contain “loops” which clearly violate the upward property of our drawing.

Therefore we will introduce static and dynamic *locks*, i.e., we prohibit edges to be considered during the shortest path computation. While the static locking by itself does not directly ensure feasibility for the upward insertion paths, it is necessary to make the dynamic locking strategy valid. The next lemma follows immediately:

Lemma 3 (Static Locking). *Considering the merge graph $\mathcal{M}(\Gamma)$, let Y_1 be the nodes that are dominated by y_1 , including y_1 itself; let X_1 be the nodes that dominate x_1 , including x_1 itself. A constraint feasible upward insertion path will not cross edges of \hat{U} that connect two nodes of X_1 or two nodes of Y_1 .*

Proof. Assume a constraint feasible upward insertion path p' would cross an edge that connects two nodes $n_1, n_2 \in Y_1$. Let d be the dummy node created by this crossing; it is dominated by either n_1 or n_2 . Then the merge graph $\mathcal{M}(\Gamma_1)$ would have a cycle as $y_1 \rightsquigarrow d$ and $d \rightsquigarrow y_1$ through the inserted edge e_1 . This contradicts the constraint feasibility of p' . The analogous holds if $n_1, n_2 \in X_1$. \square

For any face f in $\hat{\Gamma}$ let $a(f)$ be an edge corresponding to a crossing arc in R with shortest distance $\delta(f)$ to x^* . By construction, each face f consists of two directed paths—one on the left- and one on the right-hand side—from the source-switch to the sink-switch of f . We denote all edges of f between the source-switch and $a(f)$ as the *face-lock* $F(f)$.

Lemma 4 (Dynamic Locking). *For each $a(f)$, there exists a path q in R from x^* to a crossing arc corresponding to $a(f)$, where q has length $\delta(f)$ and uses no edge of $F(f')$ for all faces f' .*

Proof. Let f be a face with minimal $\delta(f)$ for which each $(x^* \rightarrow a(f))$ -path in R of length $\delta(f)$ contains at least one face-lock edge. Among all those shortest paths for this f , let q be the path which uses the fewest face-lock edges.

Traversing q from x^* , let f' be the first face for which q uses the first face-lock edge $b \in F(f')$. Let q' be q 's segment from x^* to b . By definition, we know that there exists a path $q'' := x^* \rightsquigarrow a(f')$ which is as long as q' . Since q' is shorter than q , we know that q'' does not use any face-lock edges. Hence we can create a path q^* from q by replacing q' with q'' : this replacement is straight-forwardly possible as q leaves f' over a non-face-lock edge. Note that q cannot leave f' over another face-lock edge, as this would contradict the minimality of q .

The existence of q^* —which is as long as q , but crosses less face-lock edges—constitutes a contradiction and thus the lemma holds. \square

4.3 Upward Edge Insertion Algorithm

Based on the above routing graph R and Lemmata 3 and 4 we can compute a feasible upward insertion path using a BFS algorithm adapted for 0/1 weights.

We start at x^* but instead of generating the whole routing graph we generate the successor nodes dynamically on the fly. Thereby we ignore all crossing arcs corresponding to the edges specified by static locking (Lemma 3). Furthermore we use Lemma 4 to forbid additional edges: whenever we visit a crossing arc for the first time leading into a face f , the corresponding edge in \hat{U} can be seen as $a(f)$, and hence we will block all face-lock edges $F(f)$ induced by this $a(f)$.

We call this algorithm MINIMALFEASIBLEINSERTIONPATH.

Lemma 5. *The algorithm MINIMALFEASIBLEINSERTIONPATH computes a minimal feasible upward insertion path p_1 for x_1 and y_1 .*

Proof. Assume p_1 would be infeasible. There would be a face f which cannot be drawn in an upward fashion. The routing network guarantees that a single segment of p_1 crossing through f cannot lead to such a situation. Hence there have to be at least two segments of p_1 going through f , which together contradict the upwardness of the drawing and therefore cross each other. W.l.o.g. assume that the first segment s_1 goes from some point z_l on the left of f to some point z_r on the right-hand side of f . In order to conflict with the direction of s_1 , the second crossing segment s_2 can be only one of the following:

- It starts on the left-hand side above z_l and ends on the right side below z_r .
But then we could find a shorter path which goes directly from z_l to the exit point of s_2 , removing the part of p_1 between s_1 and s_2 and all the crossings induced by it.
- It starts on the right-hand side above z_r and ends on the left side below z_l .
Due to our dynamic locking, we forbade all crossings over edges below z_l , hence s_2 cannot exit the face there.

The minimality of p_1 follows from the validity of the 0/1-BFS algorithm and Lemmata 3 and 4. \square

Corollary 2. *Let p_1 be a minimal feasible upward insertion path obtained by MINIMALFEASIBLEINSERTIONPATH, and Γ_1 the upward planar embedding arising from realizing p_1 . If the merge graph $\mathcal{M}(\Gamma_1)$ is acyclic, p_1 is a constraint minimal upward insertion path.*

There may be the situation that the computed minimal insertion path is not constraint feasible, i.e., the resulting merge graph contains a cycle. In such cases we have to resort to a heuristic for finding a constraint feasible upward insertion path which though may not be minimal:

The algorithm CONSTRAINTFEASIBLEINSERTIONPATH works similar to MINIMALFEASIBLEINSERTIONPATH but uses no dynamic locking and instead computes an *intermediate merge graph* \mathcal{I} whenever the BFS algorithm relaxes some crossing arc r : we insert “part of” e_1 along the shortest path up to r , ending at some new dummy node ξ . We then build the merge graph for this graph, adding the edge (ξ, y_1) instead of (x_1, y_1) to \mathcal{I} . If \mathcal{I} contains a cycle we know that selecting r in our current path would lead to an infeasible path, and we can forbid to use it in the BFS enumeration. Clearly, this algorithm—though always terminating—will in general not give the optimal solution, as an alternative path up to the rejected edge r might have allowed us to use r and find an overall shorter path:

Lemma 6. *The algorithm CONSTRAINTFEASIBLEINSERTIONPATH computes a constraint feasible upward insertion path p_1 for x_1 and y_1 .*

Algorithm 2 gives an overview on the overall edge insertion strategy: we try to add all edges using the optimal path of MINIMALFEASIBLEINSERTIONPATH strategy. Only if there is no more edge insertable by it, we insert a not-yet inserted edge using CONSTRAINTFEASIBLEINSERTIONPATH. Afterwards we again

Algorithm 2 Reinsert all edges

Require: sT -graph $G = (V, E)$, FUPS $U = (V, A')$ with feas. upward embedding Γ

Ensure: upward planarized graph U^* of G with embedding Γ^* , inducing Γ

```

1: List  $L := A \setminus A'$ 
2:  $U^* := U, \Gamma^* := \Gamma$ 
3: while  $L$  not empty do
4:   boolean  $success := \mathbf{false}$ 
5:   for each  $e \in L$  do
6:      $p := \text{MINIMALFEASIBLEINSERTIONPATH}(e, U^*, \Gamma^*)$ 
7:      $U^\circ, \Gamma^\circ := \text{realizePath}(p, U^*, \Gamma^*)$ 
8:     if  $\mathcal{M}(\Gamma^\circ)$  acyclic then  $\triangleright p$  was constraint feasible
9:        $U^* := U^\circ, \Gamma^* := \Gamma^\circ$ 
10:       $success := \mathbf{true}$ 
11:       $L.\text{remove}(e)$ 
12:   if not  $success$  then
13:      $e := L.\text{extractRandomElement}()$ 
14:      $p := \text{CONSTRAINTFEASIBLEINSERTIONPATH}(e, U^*, \Gamma^*)$ 
15:      $U^*, \Gamma^* := \text{realizePath}(p, U^*, \Gamma^*)$ 

```

try to use the former algorithm for the remaining edges. We iterate that process unless all edges are inserted. — As we will see in the next section, the heuristic procedure `CONSTRAINTFEASIBLEINSERTIONPATH` is only used very rarely; hence in most cases all edges are inserted optimally.

5 Runtime Analysis

We conclude the theoretic description by analyzing the algorithms' runtimes.

Lemma 7. *Let $G = (V, A)$ be an sT -graph. Algorithm 1 computes a FUPS U with a feasible embedding Γ in $\mathcal{O}(|A|^2)$ time.*

Proof. The algorithm 1 performs $|A \setminus A'| = \mathcal{O}(|A|)$ upward planarity and cycle tests. Since upward planarity testing of sT -graphs requires $\mathcal{O}(|V|)$ time [2] and cycle testing can be done in $\mathcal{O}(|A|)$, we have the above lemma. \square

Lemma 8. *Let $\bar{U} = (\bar{V}, \bar{A})$ —with embedding $\bar{\Gamma}$ —be the planarization obtained after inserting some arcs into the original FUPS U of G . Let (x, y) be the next edge to insert, and let r be the number of edges to insert afterwards.*

- (a) *Computing a minimal feasible upward insertion path for (x, y) via `MINIMALFEASIBLEINSERTIONPATH` and checking its constraint feasibility requires $\mathcal{O}(|\bar{V}| + r)$ time.*
- (b) *`CONSTRAINTFEASIBLEINSERTIONPATH` computes a constraint feasible upward insertion path for (x, y) in $\mathcal{O}(|\bar{V}|^2 + r|\bar{V}|)$ time.*

Proof. By Lemma 2, the routing network for \bar{U} with respect to $\bar{\Gamma}$ can be constructed in $\mathcal{O}(|\bar{V}|)$ time. To compute the static locks we have to construct the merge graph $\mathcal{M}(\bar{\Gamma})$ which requires $\mathcal{O}(|\bar{V}| + r)$ time. The runtime of the 0/1-BFS algorithm, including the computation of the dynamic locks, is bounded by $\mathcal{O}(|\bar{V}|)$. For the constraint feasibility checking, we have to (temporarily) insert the insertion path into \bar{U} , construct the corresponding merge graph and test for acyclicity. This can be done in $\mathcal{O}(|\bar{V}| + r)$. Thus the total runtime of (a) is dominated by $\mathcal{O}(|\bar{V}| + r)$.

The runtime analysis of `CONSTRAINTFEASIBLEINSERTIONPATH` is similar. Instead of computing the dynamic locks, we temporarily insert the current insertion path into \bar{U} after each edge relaxation, and check for acyclicity of the corresponding merge graph. Hence the runtime is $\mathcal{O}(|\bar{V}| \cdot (|\bar{V}| + r))$. \square

6 Experiments

We implemented the new level-free upward planarization approach using the open-source C++-library `OGDF` [11] and compared its performance with published results of state-of-the-art algorithms [4, 6].

In our implementation we randomize the order of the edges considered in the for-each loop (line 4 of Alg. 1) of the FUPS computation, and the order in which

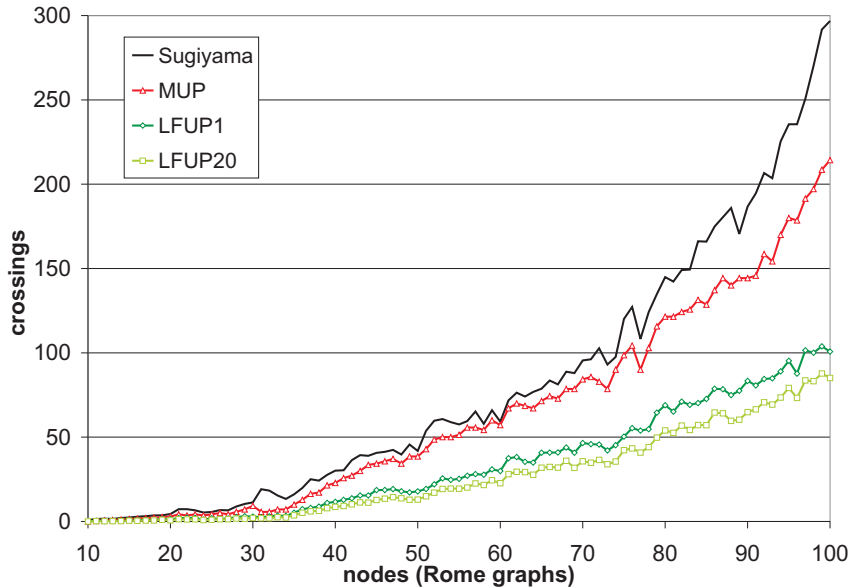


Fig. 4. Rome graphs: average crossings vs. number of nodes

edges are re-inserted (for-each loop in line 5 of Alg. 2). We denote by LFUP_i the best result obtained by after i independent calls of the algorithm. Besides comparing with published results, we also applied OGDF’s Sugiyama implementation with optimal node ranking, Barycenter heuristic, and 50 randomized runs. We used two public benchmark sets of graphs, described in the following.

Rome Graphs. The Rome graphs [5] 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—for showing the performance of the mixed-upward planarization (MUP) approach by Eiglsperger et al. [6]. In this case, all edges are directed and the graphs are acyclic; hence their approach turns into an upward planarization method.

Figure 4 shows the results for MUP, OGDF’s Sugiyama algorithm, and our new approach for 1 and 20 random calls. Each data point refers to the average number of crossings of all the graphs with the same number of nodes. The new algorithm clearly outperforms the other two approaches. Though MUP is already considerably better than the Sugiyama algorithm, LFUP_1 obtains solutions with only half as many crossings as MUP. It can also be seen that the randomization of LFUP can further reduce the number of crossings significantly.

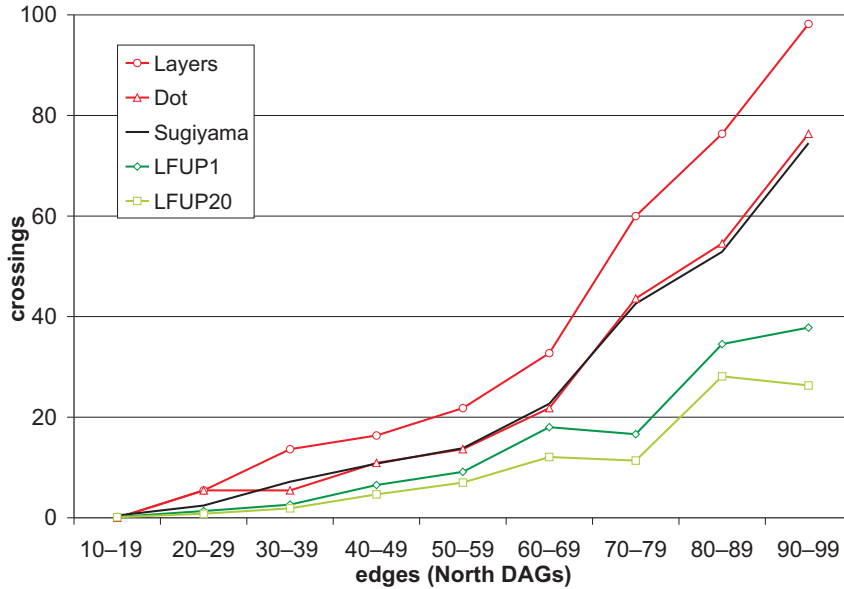


Fig. 5. North DAGs: average crossings vs. number of edges

North DAGs. The North DAGs have been introduced in an experimental comparison of algorithms for drawing DAGs [4]. They are 1158 DAGs collected by Stephen North and slightly modified by Di Battista et al. The graphs are grouped into 9 sets, where set i contains graphs with $10i$ to $10i + 9$ edges for $i = 1, \dots, 9$. From this experimental study, we used the results of the two best algorithms: *Layers* is an implementation of Sugiyama’s algorithm according to the original paper and *Dot* is a highly-optimized version of this algorithm developed by Koutsofios and North. In our diagrams, we omitted the two algorithms that use a simple method based on planarization of *st*-graphs as they perform very poorly, achieving roughly 300 crossings on average for the largest graphs.

The results are shown in Figure 5. Again, the new algorithm outperforms the three layer-based algorithms by far. While OGDF’s Sugiyama achieves almost the same results as *Dot*, LFUP1 obtains solutions with roughly half as many crossings; the multi-run version again yields significant improvements, especially for larger graphs.

Constraint feasibility. A very interesting outcome of our studies is that the minimal feasible path obtained by MINIMALFEASIBLEINSERTIONPATH is already constraint feasible in most of the cases and therefore allows us to insert the edge provably optimally.

From the 2,708,474 edge insertion calls performed by FLUP20 in total over all Rome graphs, only 114 (0.004%) require to call the CONSTRAINTFEASIBLEIN-

SERTIONPATH heuristic; this corresponds to 0.87% of the instances requiring this heuristic at all. Furthermore, the heuristic was never used for any North DAG.

Runtime. The experiments were conducted on an Intel Core-2 Duo 3.0GHz with 2GB RAM per process under Windows Vista. The maximum computation time of FLUP1 over all instances was 0.19 seconds. The large Rome graphs (90–100 nodes) take under 0.1 second on average, the large North DAGs (90–99 edges) require 60ms on average. For comparison, the runtimes of OGDF’s Sugiyama implementation were at most 15ms.

Based on our experimental comparison, we conclude that the layer-free approach achieves large improvements compared to state-of-the-art layer-based methods.

References

1. C. Batini, M. Talamo, and R. Tamassia. Computer aided layout of entity relationship diagrams. *J. Syst. Software*, 4:163–173, 1984.
2. P. Bertolazzi, G. Di Battista, C. Mannino, and R. Tamassia. Optimal upward planarity testing of single-source digraphs. *SIAM J. Comput.*, 27(1):132–169, 1998.
3. Paola Bertolazzi, Giuseppe Di Battista, Giuseppe Liotta, and Carlo Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 12(6):476–497, 1994.
4. G. Di Battista, A. Garg, G. Liotta, A. Parise, R. Tamassia, E. Tassinari, F. Vargiu, and L. Vismara. Drawing directed acyclic graphs: An experimental study. *Int. J. Comput. Geom. Appl.*, 10(6):623–648, 2000.
5. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7(5-6):303–325, 1997.
6. M. Eiglsperger, M. Kaufmann, and F. Eppinger. An approach for mixed upward planarization. *J. Graph Algorithms Appl.*, 7(2):203–220, 2003.
7. M. Eiglsperger, M. Siebenhaller, and M. Kaufmann. An efficient implementation of Sugiyama’s algorithm for layered graph drawing. In *Proc. Graph Drawing 04*, volume 3383 of *LNCS*, pages 155–166, 2004.
8. E. Gansner, E. Koutsofios, S. North, and K.-P. Vo. A technique for drawing directed graphs. *Software Pract. Exper.*, 19(3):214–229, 1993.
9. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001.
10. C. Gutwenger and P. Mutzel. An experimental study of crossing minimization heuristics. In *Proc. Graph Drawing 03*, volume 2912 of *LNCS*, pages 13–24, 2004.
11. OGDF – the Open Graph Drawing Framework. Technical University of Dortmund, Chair of Algorithm Engineering; see <http://www.ogdf.net>.
12. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Sys. Man. Cyb.*, 11(2):109–125, 1981.