Journal of Graph Algorithms and Applications
http://jgaa.info/
vol. 7, no. 2, pp. 203–220 (2003)

An Approach for Mixed Upward Planarization

Markus Eiglsperger *Frank Eppinger* *Michael Kaufmann*
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen,
Sand 13, 72076 Tübingen, Germany,
eiglsper@informatik.uni-tuebingen.de
eppinger@informatik.uni-tuebingen.de
mk@informatik.uni-tuebingen.de

Abstract

In this paper, we consider the problem of finding a mixed upward planarization of a mixed graph, i.e., a graph with directed and undirected edges. The problem is a generalization of the planarization problem for undirected graphs and is motivated by several applications in graph drawing. We present a heuristic approach for this problem which provides good quality and reasonable running time in practice, even for large graphs. This planarization method combined with a graph drawing algorithm for upward planar graphs can be seen as a real alternative to the well known Sugiyama algorithm.

Communicated by Giuseppe Liotta and Ioannis G. Tollis: submitted October 2001; revised December 2002.

Research supported in part by DFG-Grant Ka812/8-1

1 Introduction

Research for upward drawings of digraphs has been studied extensively in the last years. One reason is that such drawings have many applications in areas like workflow, project management and data flow.

An *upward drawing* of a digraph is a drawing such that all the edges are represented by curves monotonically increasing in the vertical direction. Note that such a drawing exists only if the digraph is acyclic.

A straightforward generalization of upward drawings are *mixed upward drawings*. In mixed upward drawings, only a part of the edges in the graph are directed and must point upward. Note that such a drawing exists only if the directed part of the graph is acyclic.

Mixed drawings arise in applications where the edges of the graph can be partitioned into a set which denotes structural information and a another set which does not carry structural information. An example is UML class diagrams[4] arising in software engineering. In these diagrams, the vertices of the graph represent classes in an object-oriented software system, and edges represent relations between these classes. There are two main types of relations: *generalizations* and *associations*. The generalization relations describe structural information and form a directed acyclic subgraph in the diagram. It is an often employed convention to draw generalizations upward, whereas associations can have arbitrary directions[21].

The most popular approach for creating upward drawings of digraphs is probably the Sugiyama algorithm [22]. The main idea of the Sugiyama algorithm is to assign layers to the vertices of the graph, such that edges point in ascending layer order. In a next step, the number of crossings are minimized by ordering the nodes in the layer. In the last step the nodes are assigned coordinates. For a fixed layer assignment, we call a graph *level planar* if it has a drawing which respects the layering and has no crossings. Several heuristics have been proposed for this step and used in practice, but there are also efficient algorithms to solve the level planarity problem [14][13]. There have been several attempts to apply the Sugiyama algorithm also to mixed graphs, i.e., in [21] the approach is used for UML class diagrams.

The principal step of the Sugiyama algorithm, the layer assignment, is also its most severe drawback. The layer assignment restricts the freedom of choice for the crossing minimization algorithm drastically, and there may be large differences between the number of crossings for different layer assignments of the same graph. Also, the generalizations of the Sugiyama algorithm for the mixed case have to assign layers to nodes with no directed adjacent edges. This only works when there is a low number of them, but if the directed part of the mixed graph is only small, the results are not satisfying and the layer assignment to the nodes seems artificial.

In this work we propose a new drawing strategy for upward drawings of directed graphs which is based on the concept of *upward planarity*. A directed graph is *upward planar* if it can be drawn upward without edge crossings. Our strategy consists of two phases. In the first phase, we make the input graph upward planar by replacing edge crossings by dummy nodes. We call the result of this phase *upward planarization*. In the second phase, an upward planar drawing of the upward planarization is generated and the dummy nodes are discarded.

A similar strategy has been applied very successfully in the area of drawing undirected graphs. The most popular algorithms based on this strategy are perhaps the graph drawing algorithms descending from the topology-shape-metrics approach [2][23] for orthogonal drawings. Recently, we showed in [9] how to extend the topology-shape-metrics approach to mixed upward planar drawings of *mixed upward planar graphs*, i.e., mixed graphs that have a planar drawing in which the directed part of the graph is drawn upward. The above strategy can also be applied to this algorithm, the first phase then consisting of finding a mixed upward planarization. An alternative way to draw directed graphs using the topology-shape-metrics approach is based on the concept of *quasi-upward planarity*, which is introduced in [19]. However the algorithm does not guarantee that all edges of a directed graph point upward when the graph is not upward planar.

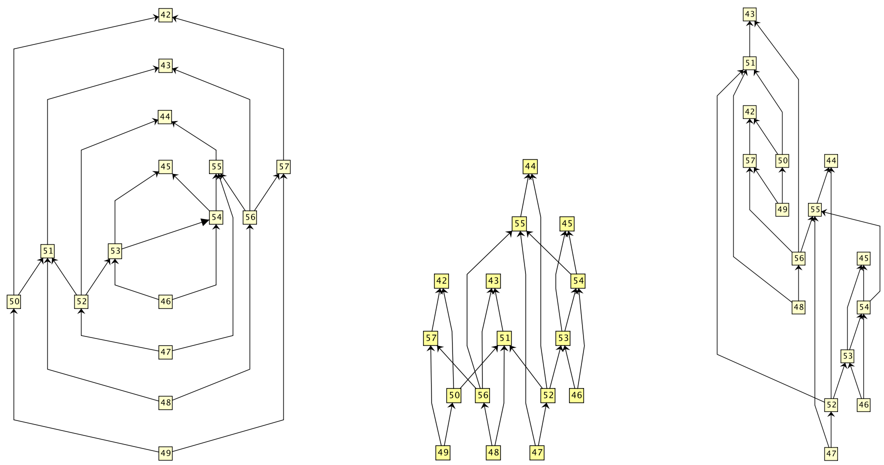
In the remainder of this paper, we concentrate on the first phase of the topology-shape-metrics approach, see [6] for a survey on graph drawing algorithms for upward planar graphs. We give an efficient heuristics that computes a high quality upward planarization of a directed graph. We concentrate on a heuristic approach, since the upward planarity test problem is already NP-complete. To our knowledge, this is the first time that this problem has been studied; work on planarization has been restricted to the undirected case until now. We also give a generalization of our algorithm for mixed graphs.

We want to emphasize that the topology-shape-metrics approach above is only one possible application of the new algorithm. Our approach also deserves attention as a stand-alone product which might be applicable in other environments.

The rest of the paper is organized as follows. Section 2 gives the formal definitions of the upward and the mixed upward planarization problem. In Section 3, we present an algorithm which solves the upward planarization problem. In Section 4, we generalize the results of Section 3 to the mixed case. Section 5 contains the results of empirical experiments performed with our algorithm. Finally, Section 6 concludes this work.

2 Upward and Mixed Upward Planarization Problem

A drawing of a graph (digraph) is a mapping of its nodes to points in the plane and of its edges to open jordan curves. A graph (digraph) is *planar* if it has a drawing where no two edges have a common point. An *upward drawing* of a digraph is a drawing such that all the edges are represented by curves monotonically increasing in the vertical direction. A digraph is *upward planar*



(a) (b) (c)

Figure 1: Three upward drawings of a directed graph. The graph is upward planar and can therefore be drawn without crossings (a). The Sugiyama-approach produces seven crossings (b) whereas our new method produces only two crossings (c).

if it has a drawing which is upward and planar at the same time. Please note that there are graphs which have an upward drawing and also have a planar drawing but do not have an upward planar drawing. An *embedding* of a graph is defined as a *cyclic ordering* of the adjacent edges of each vertex of the graph. An embedding is *planar* if there is a planar drawing of the graph which preserves this ordering. An *upward embedding* of a graph is a *linear ordering* of the adjacent edges of each vertex of the graph in which the incoming and outgoing edges form an interval. An upward embedding is *planar* if there is an upward planar drawing of the graph which preserves the corresponding ordering. Preserving the ordering means that the linear ordering is equivalent to the ordering that can be obtained by ordering the edges according to the angle they form with a ray leaving the vertex in direction of the negative x-axis. We assume in the remainder of the paper that graphs have no multiple edges and selfloops.

Given a directed graph $G = (V, E)$, the graph $G' = (V \cup V', E')$ is an *upward planarization* of G with crossing number $|V'|$ if and only if

- G' is upward planar,
- $deg(v) = 4$ for all $v \in V'$, and
- $\forall e = (v, w) \in E$, there is a path $p(e) = (v_0, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n)$ in G' with $v = v_0$, $w = v_n$ and $v_i \in V', 0 < i < n$. Every edge in E' is contained in such a path, and two paths have no edge in common.

A mixed graph is a three-tuple $G = (V, E_d, E_u) \subseteq (V, V \times V, V \times V)$, where

V is the set of vertices, E_d is the set of directed edges and E_u is the set of undirected edges.

The mixed graph $G = (V, E_d, E_u)$ is *mixed upward planar* if there is a planar drawing of G where each edge in E_d is represented by a curve monotonically increasing in the vertical direction.

A mixed upward embedding is *planar* if there is a mixed upward planar drawing of the graph which preserves the corresponding ordering.

Determining for a (mixed) graph G a (mixed) upward planarized graph is called the *(mixed) upward planarization problem.* Determining for a (mixed) graph G a (mixed) upward planarized graph with minimal crossing number is called the *(mixed) upward crossing minimization problem.*

Because the decision problem whether a graph has an upward embedding is a special case of the upward crossing minimization problem and the directed case is a special case of the mixed case, it follows that:

Corollary 1 ([10]) *The upward crossing minimization problem and the mixed upward crossing minimization problem are both NP-hard.*

3 Upward Planarization

In this section, we propose an algorithmic framework for the upward crossing minimization problem. This framework is derived from techniques for the planarization of undirected graphs, see i.e. [6].

The framework consists of three parts:

1. Construct an upward planar subgraph.
2. Determine an upward embedding of this subgraph.
3. Insert the edges not contained in the subgraph, one by one.

In the first step, a subgraph of the input graph is calculated which is upward planar. For this subgraph, an upward embedding is determined in the second step. Of course, these two steps are only conceptually separated and can be combined to one step. Note that finding a maximum upward planar subgraph, i.e., finding an upward planar subgraph with the maximum number of edges, is NP-hard. In the third step, the edges which are not part of the upward planar subgraph are inserted incrementally into the embedding. Additionally, we can perform some local optimizations on the resulting planarization to improve the quality of it.

3.1 Maximum Upward Planar Subgraph

The maximum upward planar subgraph problem can be stated as follows: Given a directed graph $G = (V, E)$. Find $E' \subseteq E$ such that the directed graph $G = (V, E)$ is upward planar with maximum number of edges.

The maximum planar subgraph problem is a related problem and a lot of algorithms have been proposed for its solution [11],[12],[17],[15],[20]. All of them, except [15], which can also compute the optimal solution when no time limit is specified, are heuristics, since the problem is NP-complete. Cimikowski [5] compared some of them empirically. In his comparison, the algorithm of Jünger and Mutzel(JM)[15] performed best in solution quality, followed by the algorithm of Goldschmidt and Takvorian (GT)[11]. The fastest algorithm was the one based on PQ-trees[11], but its performance in terms of the solution quality was significantly lower than JM and GT. In [20] Resende and Ribero give a randomized formulation of GT and show on the same test set as [5] that their formulation achieves better results with the same running time performance, except for one family of graphs where JM performs better.

However, the algorithm of GT is much easier to implement in contrast to the algorithm of JM. JM is a branch-and-cut algorithm and is, therefore, based on sophisticated algorithms for linear programming.

Because of its performance and its implementation issues, we use GT as a starting point. In the next section, we review the GT algorithm and show in the following section how it can be modified to calculate upward planar embeddings.

3.2 The Goldschmidt/Takvorian Planarization Algorithm

In this section, we review the main components of GT, the two-phase heuristics of Goldschmidt and Takvorian[11]. Our description follows the one in [20]. The first phase of GT consists in devising an ordering Π of the set of vertices of V of the input graph G. This ordering should possibly infer a Hamiltonian path. The vertices of G are placed on a vertical line according to the ordering Π obtained in the first phase, such that as many edges as possible between adjacent vertices can also be placed on the line. All other edges are drawn as arcs either right or left of the line.

The second phase of GT partitions the edge set E of G into subsets $\mathcal{L}$ (left of the line), $\mathcal{R}$ (right of the line), and $\mathcal{B}$ (the remainder) in such a way that $|\mathcal{L} + \mathcal{R}|$ is large (ideally maximum) and that no two edges both in $\mathcal{L}$ or both in $\mathcal{R}$ cross with respect to the sequence Π devised in the first phase.

Let $\pi(v)$ denote the relative position of vertex $v \in V$ within vertex sequence Π. Furthermore, let $e_1 = (a, b)$ and $e_2 = (c, d)$ be two edges of G, such that, without loss of generality, $\pi(a) < \pi(b)$ and $\pi(c) < \pi(d)$. These edges are said to *cross* if, with respect to sequence Π, $\pi(a) < \pi(c) < \pi(b) < \pi(d)$ or $\pi(c) < \pi(a) < \pi(d) < \pi(b)$.

The *conflict graph* has a vertex for every edge in G and two vertices are adjacent if the corresponding edges cross with respect to Π. It follows directly from its definition that the conflict graph is an *overlap graph*, i.e. a graph whose vertices can be represented as intervals, and two vertices are adjacent if and only if the corresponding intervals intersect but none of the two is contained by the other.

An induced bipartite subgraph of the conflict graph represents a valid assignment of the edges in G to the sets $\mathcal{L}$,$\mathcal{R}$ and $\mathcal{B}$. Since finding a maximal induced bipartite subgraph is NP-complete, even for overlap graphs, GT uses a heuristics. This heuristics calculates two disjoint independent sets of the conflict graph which, together, are a bipartite subgraph of the conflict graph.

A maximum independent set of an overlap graph can be calculated in time $O(NM)$, where N is the number of different interval endpoints and M is the number of edges in the overlap graph with the algorithm of Asano, Imai and Mukaiyama[1]. In our setting, $N \leq n$ and $M = m$, which leads to a running time of $O(nm)$.

3.3 The directed version of the GT algorithm

We now present our variant of the GT algorithm for planar upward subgraph calculation. In order to change the GT algorithm to get an upward planar subgraph, we have to modify the first step of GT, the construction of the vertex order. The vertex order must ensure that no directed edge has a target vertex which is in the order before the source vertex. This is achieved by using algorithm *vertex order* as a first phase of GT. We call this variant *directed GT* or, shorter, DGT to distinguish it from the original formulation. Algorithm 1, *directed vertex order*, is a modification of the algorithm [11]. It is a variation of a standard topological sorting algorithm and tries to maximize the number of edges between consecutive vertices in the ordering. It has been shown in [20] that this improves the quality of the result of GT. The ordering is constructed incrementally. Assume that vertex v is the vertex chosen in the previous step. The algorithm chooses a vertex in the next step which is adjacent to v, but which is not the successor of an unchosen vertex. If this is not possible, it takes a vertex of minimal degree which, additionally, is not the successor of an unchosen vertex. As the first vertex, it chooses a vertex with no incoming edge with minimal degree.

Lemma 1 *Let G be a directed graph. If the vertex order Π in the first phase of the GT algorithm is a topological order of G, the result of GT is an upward planar subgraph of G.*

Proof: Placing the vertices on a vertical line according to the ordering used by GT and drawing the edges in $\mathcal{L}$ as arcs on the left side of the line and the edges in $\mathcal{R}$ on the right side of the line yields an upward planar drawing of the subgraph calculated by GT. □

Lemma 2 *The vertex order calculated by algorithm* vertex order *is a topological order of G.*

Proof: The algorithm *vertex order* increments in each iteration the current ordering by a vertex with indegree zero. This is similar to a folklore topological sorting algorithm, see, i.e., [18] for details. □

From the sets $\mathcal{L}$ and $\mathcal{R}$ and the permutation Π, we can now easily obtain the upward planar embedding: For each node $v \in V$ we sort the edges with source

---

Algorithm 1: vertex order

Input: A directed graph $G = (V, E)$
Output: A permutation Π on the vertices
Select v_1 from G with zero indegree and minimal outdegree;
$\mathcal{V} = V \setminus \{v_1\}$;
G_1 = directed graph induced on G by $\mathcal{V}$;
for $k = 2, \ldots, |V|$ **do**
  $\mathcal{U} = \{v \in \mathcal{V} | \text{indegree}(v) = 0 \text{ in } G_k\}$;
  if *v_{k-1} is connected to a vertex in $\mathcal{U}$* **then**
    select v_k as vertex in $\mathcal{U}$ adjacent to v_{k-1} with min. degree in G_{k-1}
  else
    select v_k as vertex in $\mathcal{U}$ with min. degree in G_{k-1}
  end
  $\mathcal{V} = \mathcal{V} \setminus v_k$;
  G_k = directed graph induced on G by $\mathcal{V}$;
end
return $\Pi = (v_1, v_2, \ldots, v_{|V|})$

---

v in $\mathcal{L}$ decreasing according to Π and the edges with source v in $\mathcal{R}$ increasing according to Π and concatenate these two ordered list to one. For the incoming edges, we first sort the edges with target v in $\mathcal{R}$ decreasing according to Π and the edges with source v in $\mathcal{L}$ increasing according to Π and append the result to the list of outgoing edges.

We conclude the section with the following theorem:

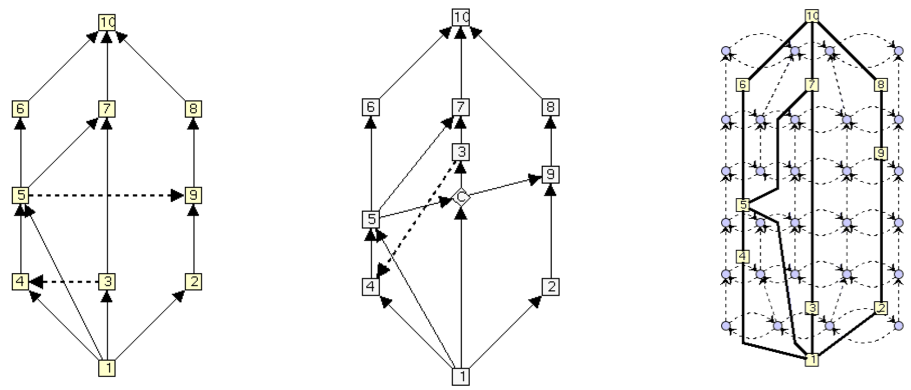
Theorem 1 *Algorithm DGT computes an upward planar subgraph, together with an upward planar embedding of this subgraph, in time $O(nm)$.*

3.4 Edge Insertion

There is an interesting difference between the insertion of directed and undirected edges. In the undirected case, the edges which are not part of the planar subgraph in the first step can be inserted independently of each other. This is different in the directed case. Here, we cannot insert an edge into the drawing without looking at the remaining edges which have to be inserted later. The reason for this is that introducing dummy nodes in the graph introduces changes in the ordering of the vertices of the graph. This may introduce directed cycles if an edge is added later. (See Figure 2).

Assume that the dashed edges have to be inserted in Fig. 2(a), and we start by inserting edge $(5, 9)$. When we do not work carefully and insert edge $(5, 9)$ as in Fig. 2(b), we produce a crossing C with edge $(1, 3)$ and some new edges, where C is involved. Then, it is no longer possible to introduce edge $(3, 4)$ without destroying the upwardness property because of the new directed cycle $5 - C - 3 - 4 - 5$.

We call a vertex with indegree 0 a *source*, and a vertex with outdegree 0 a



(a) (b) (c)

Figure 2: Edge insertion: Critical configuration and the routing graph

sink. A directed acyclic graph is called an *s-t graph* if it has exactly one sink and one source. We first restrict ourselves to s-t graphs. We show later how we can remove this restriction.

As shown above, we have to avoid cycles when we insert edges. We avoid this by *layering* the graph. A valid layering l of a directed graph $G = (V, E)$ is a mapping of V to integers such that $l(v) > l(u)$ for each edge $(u, v) \in E$. We then construct a routing graph R. The routing graph contains, for each face f and for each layer that f spans, a vertex. Two vertices lying in neighboring layers and representing the same face are connected by a directed edge of weight 0 in increasing layer order. Additionally, two vertices at the same layer i of adjacent faces are connected by an edge of weight 1 if the source vertex of an edge separating these two faces is less than or equal to i and the layer of the target node is greater than i.

In this graph, there are no edges in decreasing layer order. Each edge of weight 1 represents one crossing. A shortest path in the routing graph represents, therefore, an insertion of an edge with minimal number of crossings with respect to the given layering. Figure 2(c) shows an example for a routing graph.

Let $s(f)$, resp. $t(f)$, denote the source, resp. sink, of a face f. Note that in an s-t graph, every face has exactly one source and one sink. Furthermore, $lf(e)$, resp. $rf(e)$, denotes the face on the left, resp., right side of e. We consider the outer face as two faces, the *left-outer face*, resp. the *right-outer face*, which denote the left, resp. right part, of the outer face. Algorithm 2, *directed edge insertion*, summarizes the construction. It takes as input our current upward planar graph G, the set of remaining edges F and one edge $e \in F$. The output G' is a planarization of G and e. It uses the subroutine subdivide(G, e) which splits an edge $e = (a, b)$ into two edges $(a, w), (w, b)$, adds the vertex w to G and returns the created vertex w.

Lemma 3 *The graph G' calculated in* directed edge insertion *is an s-t graph and upward planar.*

Algorithm 2: directed edge insertion

Input: Embedded upward planar s-t graph $G = (V, E)$, $F \subseteq V \times V$, $e = (a, b) \in F$
Output: Embedded upward planarized s-t graph G' of $G = (V, E \cup e)$
calculate faces from embedding;
determine valid layering l of $(V, E \cup F)$;
Let R be an empty directed graph;
for *every face* f *of* G **do**
  create vertices $v(f, i)$ for $l(s(f)) \leq i < l(t(f))$ in R;
  for $l(s(f)) < i < l(t(f))$ **do**
    create edge of weight 0 from $v(f, i-1)$ to $v(f, i)$ in R;
  end
end
for *every edge* $e' = (c, d)$ *of* E **do**
  for $l(c) \leq i < l(d)$ **do**
    create edge of weight 1 from $v(rf(e'), i)$ to $v(lf(e'), i)$ in R;
    create edge of weight 1 from $v(lf(e'), i)$ to $v(rf(e'), i)$ in R;
  end
end
create vertex $v(a)$ and $v(b)$ in R representing a resp. b;
Insert edge of weight 0 from $v(a)$ to $v(f, l(a))$ in R if f is adjacent to a and such a vertex exists;
Insert edge of weight 0 from $v(b)$ to $v(f, l(b) - 1)$ in R if f is adjacent to b and such a vertex exists;
Calculate shortest path p from $v(a)$ to $v(b)$ in R;
$E' = E$,$V' = V$,$G' = (V', E')$;
Let $e_0, \ldots, e_n$ be the edges of weight 1 in p;
for $0 \leq i \leq n$ **do**
  w_i =subdivide(G', e_i);
end
Add an edge between a and w_0, w_n and b, and w_i and w_{i+1} in E';
return G'

Proof: In the edge insertion step, we do not decrease the indegree or the outdegree of any vertex existing already in the input graph. Therefore, we only have to show that none of the inserted vertices is a sink or a source. But this is true, since each of these vertices has indegree two and outdegree two. G' is upward planar, since there are no crossings and the layering is observed. □

Lemma 4 *The graph* $(V', E' \cup F \setminus e)$ *is acyclic.*

Proof: Assume that there is a directed cycle. Each inserted vertex w_i is induced by an edge of weight 1 in the routing graph which connects two face vertices lying in the same layer. Assign this layer to node w_i. Then, each vertex has a layer assigned, and there are no edges which point in decreasing layer order. Thus, the cycle can only contain vertices in the same layer. These can only be vertices w_i by the construction of the layering. But, from this fact it follows that the shortest path had a directed cycle which is a contradiction. □

Lemma 5 *Algorithm* directed edge insertion *has time complexity* $O(|V|^2)$,

Proof: The faces of the graph can be computed in linear time from the embedding. A valid layering with a minimal number of layers can also be computed in linear time using a topological sorting. The maximum number of layers is linear, since a topological sorting is an upper bound for the number of layers. Hence, the number of vertices in the routing graph is $O(|V|^2)$ and, since each vertex has constant degree, the total size of the routing graph is $O(|V|^2)$. Because the maximum cost of an edge is 1, we can use Dial's shortest path algorithm[8] which has linear running time in this case. The insertion of the edge can clearly be done in linear time. □

The following theorem summarizes the lemmas above.

Theorem 2 *Algorithm* directed edge insertion *inserts one edge in an embedded upward planar s-t graph* $G = (V, E)$ *in time* $O(|V|^2)$ *without introducing cycles with a set of not yet inserted edges. The planarized graph is an s-t graph.*

3.5 The Complete Algorithm

Algorithm 3, *upward planarization*, contains a description of the algorithm. Note that if the input graph for the second phase is not an s-t graph, we augment it to a planar upward s-t graph, see [3] for a linear time algorithm. Edges in the routing graph representing an edge added in the augmenting step are assigned weight 0, because they do not introduce a real crossing. The removed edges are inserted in random order in the second phase. After the routing, the augmenting edges are removed. Note that the augmentation does not affect the worst-case running time of the algorithm, since the number of edges in the graph remains linear in the number of nodes.

Algorithm 3: upward-planarization

```
calculate embedded mixed upward planar subgraph with DGT;
augment subgraph to an s-t graph;
for Each removed directed edge do
    call algorithm directed edge insertion;
end
remove edges inserted in augmentation process;
```

From the discussion above, we derive the following theorem:

Theorem 3 *Let* $G = (V, E)$ *be a directed graph. Algorithm 3,* upward-planarization, *creates an embedded upward planarized graph of G in time* $O(|V||E| + (|V| + c)^2|E|)$, *where c is the number of crossings of the planarized graph. When G is sparse, i.e.* $|E| = O(|V|)$, *the algorithm* upward-planarization *runs in time* $O((|V| + c)^2|V|)$.

However, the time bound in the theorem above is very pessimistic. In our experiments, the running time of the algorithm is reasonable, even for larger graphs.

3.6 Rerouting

In this section, we present a local optimization method for an upward planarization. One step of the method removes a path representing an edge from the planarization, and tries to reinsert it with fewer crossings. To test whether it can be reinserted with fewer crossings we first augment the graph to an s-t-graph after the removal of the path. Then we construct from this s-t graph the routing graph. Testing whether the edge can be inserted with fewer crossings reduces again to a shortest path problem in the routing graph. If we succeed, we change the planarization according to the new routing, otherwise, we do not change the planarization. In any case, the augmented edges are removed. We iterate this local optimization until we either do not make any further improvements, i.e., there is no edge for which we can find a routing with less crossings. This is realized by defining a set of edges $Cand$ which contains all edges of the original graph which have crossings in the planarization. In each iteration we randomly choose one edge and perform the local optimization step for the path defined by this edge. If the planarization had been improved we recalculate $Cand$ and start again. We stop when $Cand$ is empty. Since the local optimization is time consuming, the total number of local optimizations steps can be bounded by a constant.

4 Mixed Upward Planarization

In this section we show how the concepts in the previous sections can be extended to the mixed case, i.e., the input graph is a mixed graph.

For the mixed planar subgraph calculation, we also use the GT algorithm. As in the upward case, we only have to take care of the vertex ordering. We use Algorithm 4, *mixed vertex order*, a modified version of Algorithm *vertex-order* for this, which ignores the direction of the undirected edges.

The modifications follow the intuition that the undirected edges allow more freedom since we can choose the direction. So, the idea is to prefer the directed edges when computing the planar subgraph. One variant of the planar subgraph algorithm that takes this aspect into account, is to extend the GT approach by assigning different weights to the directed and undirected edges and then optimize over the weighted sum of the edges [1]. The actual choice of the weights depends on the application as well as on the class of graphs to consider and is the subject of further research.

The upward edge insertion algorithm can be extended to the mixed case by directing the undirected edges in G temporarily according to the ordering in GT. We then insert the removed directed edges iteratively in the graph as described in section 3.4. Next we undirect the temporarily directed edges. Finally we

Algorithm 4: mixed vertex order

Input: A mixed graph $G = (V, E, F)$
Output: A permutation Π on the vertices
Select v_1 from G with zero indegree and minimal degree;
$\mathcal{V} = V \setminus \{v_1\}$;
G_1 = mixed graph induced on G by $\mathcal{V}$;
for $k = 2, \ldots, |V|$ **do**
  $\mathcal{U} = \{v \in \mathcal{V} | \text{indegree}(v) = 0 \text{ in } G_k\}$;
  if v_{k-1} *is connected to a vertex in* $\mathcal{U}$ **then**
    select v_k as vertex in $\mathcal{U}$ adjacent to v_{k-1} with min. degree in G_{k-1}
  else
    select v_k as vertex in $\mathcal{U}$ with min. degree in G_{k-1}
  end
  $\mathcal{V} = \mathcal{V} \setminus v_k$;
  G_k = mixed graph induced on G by $\mathcal{V}$;
end
return $\Pi = (v_1, v_2, \ldots, v_{|V|})$

insert the removed undirected edges by an standard edge insertion algorithm for undirected graphs [6]. Algorithm 5, *mixed-upward-planarization*, summarizes this.

Algorithm 5: mixed-upward-planarization

calculate embedded mixed upward planar subgraph with GT;
direct undirected edges in the subgraph temporarily;
augment subgraph to an s-t graph;
for *Each removed directed edge* **do**
  call algorithm directed edge insertion;
end
remove edges inserted in augmentation process;
undirect edges which have been directed;
for *Each removed undirected edge* **do**
  call algorithm undirected edge insertion;
end

Theorem 4 *Let $G = (V, E_d, E_u)$ be a mixed graph. Algorithm* mixed-upward-planarization *creates an embedded mixed upward planarized graph of G in time $O(|V|(|E_d| + |E_u|) + (|V| + c)^2|E_d| + (|V| + c)|E_u|)$, where c is the number of crossings in the planarized graph.*

5 Experiments

In this section we present the results of an experimental comparison of our algorithm with the Sugiyama algorithm for directed acyclic graphs.

All experiments have been performed on a Pentium IV System with 1.8 GHz and 512 Megabyte main-memory running Linux. We implemented our algorithm in pure JAVA based on the yFiles library [24]. For the experiments we used a randomized version of GT which takes the largest subgraph from 150 different node orderings. In the experiments we did not use rerouting. We compare our algorithm to the Sugiyama implementation in yFiles [24]. This implementation uses a randomized version of the iterated barycenter method. This method was the clear winner of an experimental comparison of heuristics for the crossing minimization problem of layered graph [16]. All experiments have been performed using JDK 1.4.1.

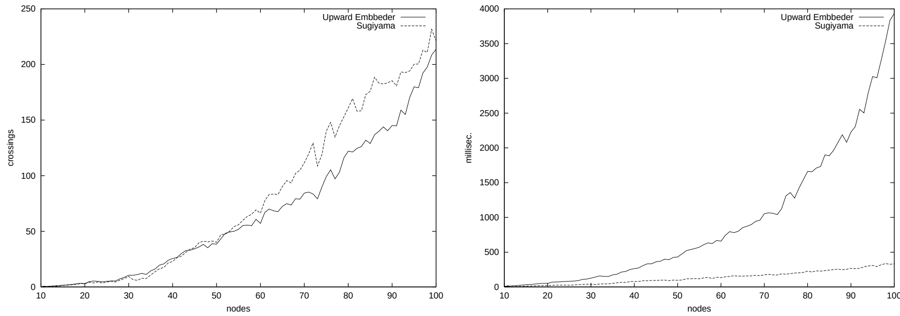
We performed our experiments on three test sets:

Rome Graphs. The Rome-graphs test suite[7] contains about 10.000 undirected connected graphs. The number of nodes in the test suite ranges from 10 to 100, the average density of the graphs ranging from 1 to 2 with average value of 1.3. We transformed the undirected graphs to directed acyclic graphs by directing the edges according to an ordering of the nodes of the graph. As ordering we chose the implicit ordering of the nodes as defined in the file.

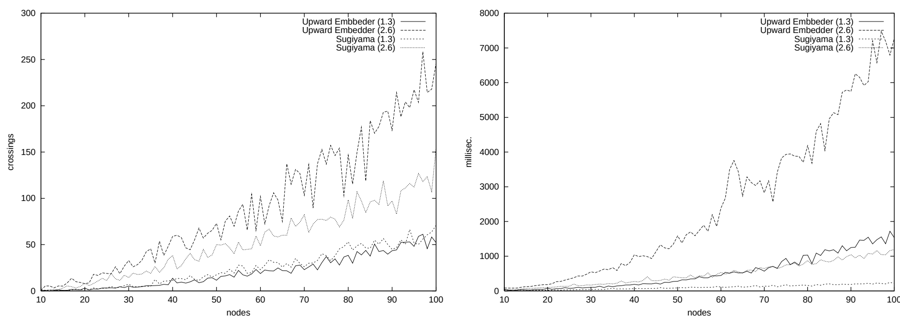
Upward Planar Graphs. There are two test sets consisting of connected upward planar graphs, the first contains graphs with density 1.3, the second graphs with density 2.6. Both test sets contain 910 upward planar graphs, the number of nodes ranging in each form 10 to 100, containing 10 graphs for each node count. The graphs were generated the following way: First a random set of points in a triangle was generated. For this point set a Delaunay triangulation was performed which yields a planar triangulated graph. We deleted edges randomly until we reached the desired density. To assure that the generated graphs were connected we computed a spanning tree of the triangulated graph by randomized DFS and ensured that edges in the spanning tree are not deleted in the previous step. Finally we directed the edges according to the coordinates of their endpoints.

Graphs With Limited Height. There are two test sets which contain connected directed graphs with maximum height three, the first with density 1.3, the second with density 2.6. Maximum height three means in this setting that they have a layer assignment with at most three layers. Both test sets contain 910 upward planar graphs, the number of nodes ranging in each form 10 to 100, containing 10 graphs for each node count. The graphs were generated the following way: First we distributed randomly the nodes in three layers. To assure that a generated graph was connected we generated a spanning tree for it. Then we inserted edges randomly between nodes in neighbored layers until the desired density was reached.

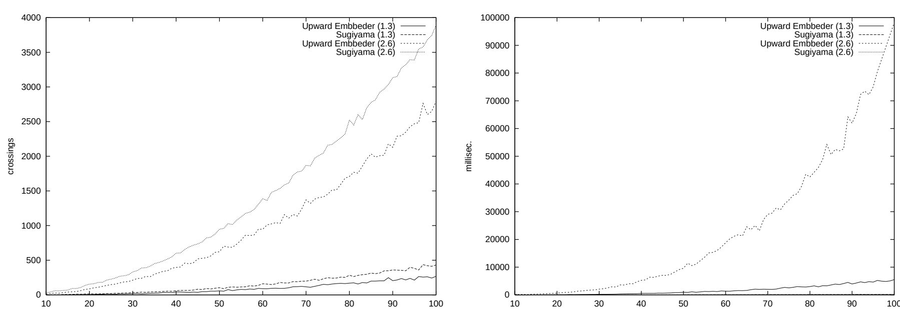
Figure 3 shows the results of our experiments. The left diagrams show the relation between the average number of crossings and the number of vertices. The right diagrams show the relation between average running time and the



(a) Rome Graphs



(b) Upward Planar Graphs



(c) Limited Height Graphs

Figure 3: Results of experiments: Number of crossings and running time in milliseconds.

number of vertices. For the test sets with density 1.3 our algorithm yields better results than the Sugiyama approach. In the case for limited height graphs, the improvements are drastic. For the test sets with density 2.6 the Sugiyama approach is the clear winner. In terms of running time, the Sugiyama approach clearly outperforms our approach, however the running time of our algorithm is still acceptable for interactive use.

6 Conclusion

In this paper, we gave a new algorithm for the problem of finding a upward planarization for graphs with directed and undirected edges as well. Our approach generalizes the related problem for undirected graph and emphasizes on the practical needs for such methods, namely practical efficiency and good quality, even for large graphs. The concept is designed so flexible that many additional requirements like constraints or interactivity might be incorporated. Hence, together with a graph drawing algorithm for upward planar graphs it can be viewed as a reasonable alternative to the well known Sugiyama algorithm.

Acknowledgments

The authors wish to thank the referees for their useful suggestions.

References

[1] T. Asano, H. Imai, and A. Mukaiyama. Finding a maximum weight independent set of a circle graph. *IEICE Tranactions*, E74:681–683, 1991.

[2] C. Batini, E Nardelli, and R. Tamassia. A layout algorithm for data-flow diagrams. *IEEE Trans. Softw. Eng.*, SE-12(4):538–546, 1986.

[3] P. Bertolazzi, G. Di Battista, G. Liotta, and C. Mannino. Upward drawings of triconnected digraphs. *Algorithmica*, 6(12):476–497, 1996.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language*. Addison-Weslay, 1999.

[5] R. Cimikowski. An analysis of heuristics for the maximum planar subgraph problem. In *Proceedings of the 6th ACM-SIAM Symposium of Discrete Algorithms*, pages 322–331, 1995.

[6] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[7] G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algortihms. *Comput. Geom. Theory Appl.*, 7:303–325, 1997.

[8] R. Dial. Algorithm 360: Shortest path forest with topological ordering. *Communications of ACM*, 12:632–633, 1969.

[9] M. Eiglsperger, U. Foessmeier, and M .Kaufmann. Orthogonal graph drawing with constraints. In *Proc. 11th ACM-SIAM Symposium on Discrete Algorithms*, pages 3–11, 2000.

[10] A. Garg and R. Tamassia. On the complexity of upward and rectilinear planarity testing. In *Proceedings of the 2nd International Symposium on Graph Drawing (GD'94)*, volume 894 of *LNCS*, pages 286–297, 1995.

[11] O. Goldschmidt and A. Takvorian. An efficient graph planarization two-phase heuristic. *Networks*, 24:69–73, 1994.

[12] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy. $o(n^2)$ algorithms for graph planarization. In *IEEE Trans. on CAD*, volume 8 (3), pages 257–267, 1989.

[13] M. Jünger and S. Leipert. Level planar embedding in linear time. In J. Kratochvil, editor, *Proceedings of the 7th International Symposium on Graph Drawing*, volume 1547, pages 72–81. Springer Verlag, 2000.

[14] M. Jünger, S. Leipert, and P. Mutzel. Level planarity testing in linear time. In *Proceedings of the 6th International Symposium on Graph Drawing (GD'98)*, volume 1547 of *LNCS*, pages 224–237, 1998.

[15] M. Jünger and P. Mutzel. Solving the maximum weight planar subgraph problem by branch & cut. In *Proc. of the 3rd conference on integer programming and combinatorial optimization (IPCO)*, pages 479–492, 1993.

[16] M. Jünger and P. Mutzel. 2-layer straightline crossing minimization: Performance of exact and heuristic algorithms. *Journal of Graph Algorithms and Applications (JGAA)*, 1(1):1–25, 1997.

[17] G. Kant. An $O(n^2)$ maximal planarization algorithm based on PQ-trees. Technical Report RUU-CS-92-03, CS Dept., Univ. Utrecht, Netherlands, 1992.

[18] K. Mehlhorn and S. Näher. *The* LEDA *Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[19] W. Didimo P. Bertolazzi, G. Di Battista. Quasi-upward planarity. *Algorithmica*, 32(3):474–506, 2002.

[20] M. Resende and C. Ribeiro. A grasp for graph planarization. *Networks*, 29:173–189, 1997.

[21] J. Seemann. Extending the Sugiyama algorithm for drawing UML class diagrams: Towards automatic layout of object-oriented software diagrams. In *Proceedings of the 5th International Symposium on Graph Drawing (GD'97)*, volume 1353 of *LNCS*, pages 415–424, 1997.

[22] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, February 1981.

[23] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[24] R. Wiese, M. Eiglsperger, and M. Kaufmann. yfiles: Visualization and automatic layout of graphs. In *Proceedings of the 9th International Symposium on Graph Drawing (GD'01)*, LNCS, pages 453–454. Springer, 2001.