On Bend-Minimum Orthogonal Upward Drawing of Directed Planar Graphs

Ulrich Fößmeier[1], Michael Kaufmann[1]

Universität Tübingen, Wilhelm- Schickard- Institut, Sand 13, 72076 Tübingen, Germany,
email: foessmei@informatik.uni-tuebingen.de, mk@informatik.uni-tuebingen.de

Abstract. In last year's graph drawing workshop GD'93 we considered a restricted version of the problem of minimization of bends in orthogonal upward drawings. Inserting the severe restriction that each node should have an incoming edge from below and an outgoing edge upwards (if such edges exist), we were able to get optimal bounds on the number of bends in linear time. In this paper now, we release this restriction completely. The problem becomes much harder. Starting from a fixed planar topological embedding we are able to reduce the problem to a min-cut problem and present three algorithms: a) Find an orthogonal upward drawing without any bend, if such a drawing exists (in linear time), b) find a bend-minimum solution, if the undirected version of the graph requires no bends (in time $O(n^2 \cdot \log n)$, n being the number of vertices of the graph), c) apply our technique to the general case; here we could not prove the optimality up to now. But the achieved number of bends does not exceed the optimum by more than the optimal number of bends in Tamassia's undirected case, i.e. our algorithm needs at most twice the number of bends as necessary in this case.

1 Introduction

Embedding a planar graph G into another graph H is one of the most important classes of graph drawing problems. Especially for the case where the host graph H is a rectilinear grid (i.e. vertices of G are mapped to grid points and edges of G run along grid lines), several algorithms to compute an embedding with nice properties can be found in the literature ([12, 14, 5, 10]). It is well known that the problem of finding a drawing that needs a minimum area or minimum edge lengths is NP-hard. But another important optimality criterion can be reached in polynomial time: Tamassia [12] gave an algorithm that, given a planar graph G together with a planar topological embedding computes an orthogonal layout of G with the minimum number of bends preserving the embedding. He showed that this problem can be reduced to a min-cost-flow-problem in a network. Di Battista, Liotta and Vargiu [2] found an algorithm for bend-minimum drawings for special graph classes (series-parallel graphs, 3-planar graphs) independent from the topological embedding. Storer [10] and Tamassia/Tollis [14] gave very fast heuristics but could not prove the optimality of their approaches.

Another important way to display the structure of a graph is 'upward drawing'. Directed edges are embedded such that they are monotonically increasing curves. This is a very popular way to draw graphs [9, 4, 16, 3, 5, 1, 11], but almost all the criteria become hard to achieve when preserving upwardness of the edges (crossing number [6], area [5]).

Only recently Garg and Tamassia [8] showed that finding an orthogonal planar drawing of a planar graph G that needs the minimum number of bends among all

planar topological embeddings is an NP-hard problem, even finding an approximation with an $O(n^{1-\epsilon})$ error, n being the number of vertices of G, is NP-hard. Thus, in search of efficient algorithms for orthogonal planar drawings, we consider the planar topological embedding of the graph as an input that we do not change within the algorithm.

In this paper we combine the two targets mentioned above: We draw graphs 'upward', minimizing the number of bends. Contrasting to our first attempt [7], each edge is only required to have a y-monotonic representation. Obviously we need at least as many bends as in the 'undirected' version successfully considered by Tamassia [12]. We give a test of a possible no-bend upward drawing and present our main algorithm that optimally solves the problem for graphs, which have a no-bend representation in the undirected case. For that we have to solve a min-cut problem. For the general problem, we give an approximation algorithm which consists of a mixture of the min-cost-flow-approach of Tamassia and our new min-cut technique, achieving a good bound for the general case: The number of bends does not exceed the optimum by more than the minimum number of bends in Tamassia's undirected version.

2 Definitions and Properties

Definition 1

1. *An* orthogonal planar drawing *of a planar graph G is a matching of the nodes of G to pairwise different grid points and of the edges of G to paths on the grid between the corresponding points such that these paths are pairwise disjoint and do not cross.*
2. *An* upward planar drawing *of a directed planar graph G is a matching of the nodes of G to pairwise different geometric points on the plane and of the edges of G to simple non-crossing curves between the corresponding points such that all these curves are monotonically increasing.*
3. *An* orthogonal upward planar drawing (OUPD) *of a directed planar graph G is a drawing that is orthogonal and upward, i.e. each edge of G consists of segments going to the left, upward or to the right.*
4. *Given a planar graph G drawn in the plane, an OUPD R of G is called* region preserving, *if R and G have the same planar topological embedding.*

In the rest of the paper, with 'planar graph' we mean a planar graph together with a planar topological embedding for it (we only regard region preserving drawings).

Definition 2 *A planar graph that allows an OUPD is called an* upward planar graph.

Definition 3 *A directed planar graph G is called an* s-t-graph, *if G is acyclic with exactly one source s and one sink t, and G is embedded in the plane such that s and t are lying on the boundary of the external face.*

Lemma 1

1. *A planar graph G has an orthogonal planar drawing if and only if the maximum node degree of G is at most 4.*
2. *A directed planar graph G has an upward planar drawing, if and only if G is a subgraph of an s-t-graph.*

3. *A directed planar graph G has an OUPD, if and only if G is a subgraph of an s-t-graph and its maximum node degree is at most 4.*

Proof
1.) and 2.) can be found in [12] and [4]
3.) follows from 1.) and 2.). ◇

The next lemma is taken from [13].

Lemma 2 *Let G be a planar s-t-graph embedded in the plane; the incoming edges of each node v appear consecutively around v, the same holds for the outgoing edges. For the orthogonal case this means that at a node v of degree 4 with two incoming and two outgoing edges it is not possible that the incoming and outgoing edges alternate. Moreover at a node with more than one exit (entry) it makes sense to say that one exit (entry) lies to the left or to the right of the other.*

3 How to Draw a No-Bend Upward Planar Graph

In this section we show how to compute an OUPD of an upward planar graph G with no bends, if this is possible. In other words, if we have the information that G can be drawn without any bends, we want to determine this layout.

We solve this problem with an algorithm similar to the one described in [7]: Let V and E be the nodes and edges of G, respectively. We start with edges e whose incident nodes determine the direction of e, if e should be drawn without any bends. These edges may force some other edges to have a certain direction and so on. If there should remain edges that cannot be forced to have a fixed direction, we choose one of them and give it arbitrarily a direction that creates no contradiction to other remaining edges.

More precisely: A node v having three exits (entries) determines the directions of these three incident edges. We call such a node a *bowl*. A node v having two exits (two entries analogously) restricts the possibilities for the directions of these edges: A right exit can get direction right (r) or vertical (v), a left exit left (l) or vertical. So we assign to each edge e two sets of directions (subsets of $\{r, v, l\}$), one for the 'entry-side' and one for the 'exit-side' of e. Obviously the direction of every edge must be found in the intersection I_e of its two direction sets. So we start with edges e having a set I_e of cardinality one (empty sets I_e do not exist, since we can draw the graph without bends). At the beginning there are only two types of such edges: bowls and edges e which are the right (left) of two exits of some node v and the right (left) of two entries of another node w. Such edges are called *right (left) flanks* (see Fig. 1).

Every time we fix the direction of an edge e we have to update the sets $I_{e'}$ of edges e' being incident to e; for these edges e' now may have only two possible directions or even only one (see Fig. 2). Note that all update operations together only require linear time since 'updating' always means reducing the sets I_e and these sets together have a size of at most $3 \cdot |E| = O(n)$. Some of these edges e' may get sets $I_{e'}$ of cardinality one now and are candidates for the next edge to get its direction. This step of the algorithm, where the possibilities for edge directions are reduced because of the influence of other edges, is called *propagation* of directions, and the edges where this propagation runs along build a *propagating path.*

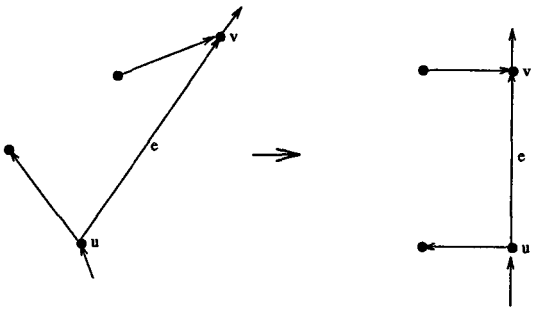

Fig. 1. A right flank

If every edge e has a set I_e of cardinality more than one, we choose arbitrarily a remaining edge e with I_e of cardinality two (if such e does not exist, choose one with I_e of cardinality three) and assign it a feasible direction.

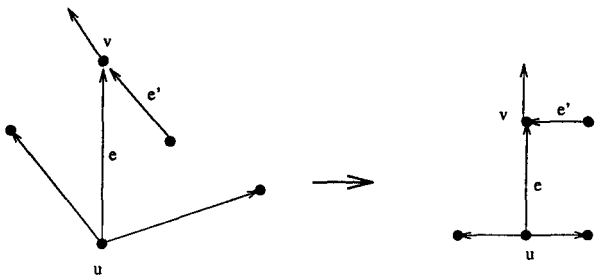

Fig. 2. The direction of the edge e' is unique

Summarizing these observations we formulate

Algorithm 1

for all $v \in V$ **do**
    compute the direction sets for all edges e being incident to v;
for all $e \in E$ **do**
    compute I_e, the intersection of the two direction sets of e;
$U := E$;
while $U \neq \emptyset$ **do**
    choose $e = (v, w) \in U$ with a set I_e of minimal cardinality;
    $U := U \backslash \{e\}$
    assign to e arbitrarily one of the directions in I_e;
    update $I_{e'}$ for all $e' \in U$ being incident to v or w;
endwhile;

Theorem 1 *Algorithm 1 computes an OUPD of a graph G without any bends, if this is possible.*

Corollary 1 *A slightly changed version of Algorithm 1 decides whether a graph G has an OUPD with no bends or not.*

Proof: If G needs a bend, then there is an edge to which Algorithm 1 would assign two different directions, i.e. a set I_e is empty. In this situation we stop and output 'no layout without bends'. ◇

Corollary 2 *Algorithm 1 runs in linear time.*

Proof: Follows from the considerations above; a more precise analysis is similar to the proof from [7]. ◇

4 Towards the General Case

Now we want to solve a larger class of problems besides upward planar graphs without any bends. We classify the bends being necessary in an OUPD into two sets: Given an upward planar graph G we consider the corresponding undirected planar graph G', having for every directed edge in G an undirected edge between the same nodes. Run Tamassia's algorithm for G' and you get a bend-minimum solution for it. The bends of this solution are called *structural bends* of G, because they are necessary because of the structure of the graph and are not forced by any upward condition. In this section we only consider graphs without structural bends, i.e. graphs that can be drawn without bends, if we allow the edges to be directed in all four directions. In the following we present an algorithm that computes an OUPD for a given upward planar graph G with the minimum number of bends, if G has no structural bends.

A bend is always necessary, if an edge $e = (u, v)$ is forced to have different directions in the course of Algorithm 1, i.e. its I_e becomes empty in the course of the algorithm. Our aim is to find the sources of such conflicts. For this purpose we run the propagation step of Algorithm 1 for every pair of determined edges e_1 and e_2 (flanks or bowls) and look whether there is edge e', where the two sets of directions assigned in this step are disjoint. We say that e_1 and e_2 are *conflicting* and the edges which propagated the directions from e_1 to e_2 build the *conflict path* between e_1 and e_2. One of the edges of the conflict path must have a bend. An example can be seen in Fig. 3a): e_1 and e_5 are flanks and should have vertical direction; all edges between e_1 and e_5 propagate the direction, so one of the edges $\{e_1, e_2, e_3, e_4, e_5\}$ has to make a bend. The conflict paths should be directed in such a way that the corresponding bend has the 90°-angle at its right side. Thus the conflict path in the example is: e_5, e_4, e_3, e_2, e_1. Note that the direction of an edge in a conflict path has nothing to do with the direction of the corresponding edge in the graph.

If two conflict paths $P_1 = e_{i_1}, \ldots, e_{i_k}$ and $P_2 = e_{j_1}, \ldots, e_{j_l}$ join at an edge e, they propagate the same direction to e. So the endpoint of P_1 is also conflicting with e_{j_1} and vice versa. Thus we have paths $e_{i_1}, \ldots, e_{i_k}$, $e_{i_1}, \ldots, e_{j_l}$, $e_{j_1}, \ldots, e_{i_k}$ and $e_{j_1}, \ldots, e_{j_l}$. There must be a bend on each of these paths. We can solve this problem considering the paths as a part of a network that we can derive from the graph G and looking for a min-cut for this network. On every edge of the min-cut we place a bend. We describe the exact definition of the network later. To realize this idea, we

have to solve some technical problems: Paths might require two bends along them. We call such paths *double paths*. An example for a double path can be seen in Fig. 3b).

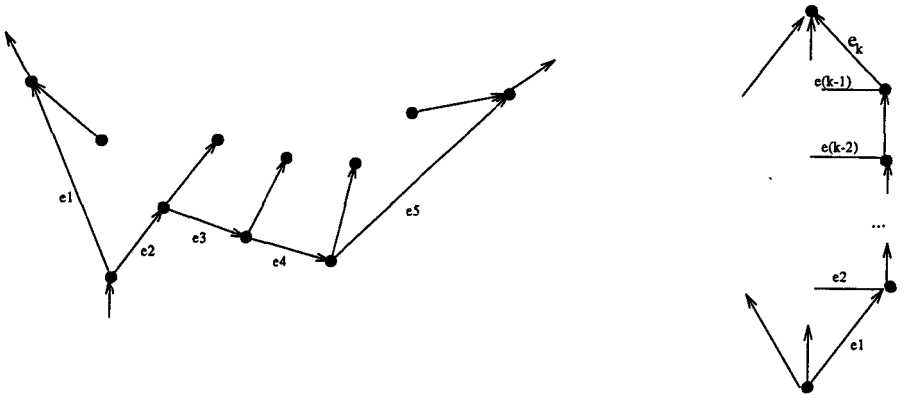

Fig. 3. a) A conflict path, b) A double path between $e1$ and ek

Note that the edges $e_2, \ldots e_{k-1}$ are necessary to propagate the directions between the bowls. Let e_r be the edge with the smallest index among them being directed to the path (in the example from left to right) and e_l the edge with the largest index among them being directed off the path; if one of them does not exist, set $e_r = e_k$ or $e_l = e_1$, respectively. We can split the double path into two simple paths $e_1, \ldots, \overline{e_r}$ and $\overline{e_l}, \ldots, e_k$, where $\overline{e_r}$ is the edge on the double path entering the same node as e_r and $\overline{e_l}$ is the edge on the double path leaving the same node as e_l; note that these simple paths are disjoint or they overlap in exactly one edge $e^* = \overline{e_r} = \overline{e_l}$. Since we need two bends between e_1 and e_k (one at each simple path), it is not sufficient to place one bend onto the edge e^*, but a feasible solution would be to place two bends there. Thus in our network the path $e_1, \ldots, \overline{e_r}$ must be finished before the path $\overline{e_l}, \ldots, e_k$ starts, even if they are overlapping. Therefore each edge of the graph corresponds to two nodes in the network and the paths start at the second copy and end at the first. For technical reasons we must insert a dummy node between these two copies to separate the two overlapping simple paths which define a double path (see below: construction of the network).

A special case arises at paths of length one, i.e. an edge e where a bend is obligatory; here this construction would not lead to a way from S to T. In this case we make an extra copy of the nodes corresponding to e and edges from S to this copy and from the copy to T. All simple paths containing e should be removed and double paths containing e should become simple paths now, because the obligatory bend on e satisfies the corresponding conflict conditions.

For an example see Fig. 4: The conflict paths are:
$P_1 = (3, 5, 12)$ $P_2 = (3, 5, 10, 6, 9)$ $P_3 = (12, 14, 18)$ $P_4 = 18, 21)$ $P_5 = (21)$

P_5 has length 1 and is part of P_4; so P_4 is removed and we get four conflict paths.

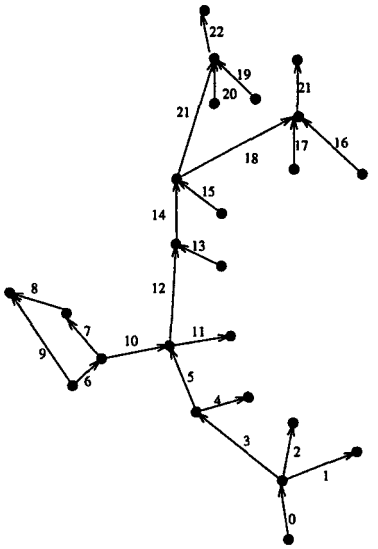

Fig. 4. Example 1

The network is constructed as follows:

Let $\hat{E} \subseteq E$ be the set of the conflict paths of length one, and $\hat{P}$ the set of conflict paths after removing and modifying paths that contain edges of $\hat{E}$.

For every edge e being part of some conflict path in $\hat{P}$ we define three nodes v_e, v'_e and v''_e in the network. For every $e \in \hat{E}$ there are the additional nodes v_e, v'_e and v''_e. In addition, the source S and the sink T should be nodes in the network.

There are five kinds of edges in the network:

- Edges of capacity ∞ from S to v''_e, if e is the first edge of a conflict path in $\hat{P}$ and from S to v_e, if $e \in \hat{E}$.
- Edges of capacity ∞ from v'_e to T, if e is the final edge of a conflict path in $\hat{P}$ and from v''_e to T, if $e \in \hat{E}$.
- Edges of capacity ∞ from v'_e to v''_e for every e being part of a conflict path in $\hat{P}$ and for every $e \in \hat{E}$.
- Edges of capacity 1 from v_e to v'_e for every e being part of a conflict path in $\hat{P}$ and for every $e \in \hat{E}$.
- Edges of capacity 1 from v''_{e_1} to v_{e_2}, if e_1 and e_2 are consecutive edges in some conflict path in $\hat{P}$.

The network resulting from the example of Fig. 4 can be seen in Fig. 5.

The proof of Theorem 2 will show that every cut of this network corresponds to a solution of the drawing problem, where there is a bend on an edge e if and only if one of the edges leaving v_e or v''_e in the network belongs to the cut, and there are two

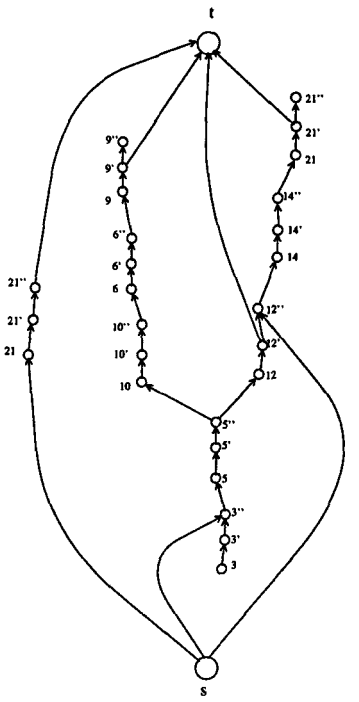

Fig. 5. The network for Example 1

bends on an edge e if and only if these two edges of the network are both belonging to the cut. Hence the minimum cut corresponds to the bend-minimum solution.

Theorem 2 *Algorithm 2 computes an OUPD with the minimum number of bends for an upward planar graph G without structural bends.*

Algorithm 2

a) Determine for every edge e of the graph G, if e is part of a conflict path in P;
b) Determine the set of edges $\hat{E} \subseteq P$ with an obligatory bend;
c) Modify the information from a), such that for every edge e is known, if e is part of a conflict path in $\hat{P}$;
d) Construct the network according to the rules described above;
e) Solve the min-cut-problem;
f) Assign bends to the edges belonging to the cut;
g) Run Algorithm 1 to determine the directions for the remaining edges.

A min-cut for Example 1 are the edges leaving the nodes $3''$, $12''$ and 21. The resulting layout is shown in Fig. 6.

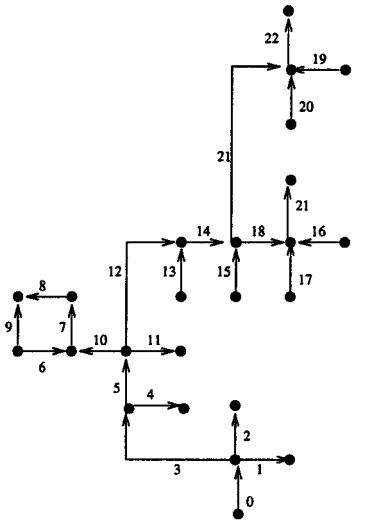

Fig. 6. The result for Example 1

To prove Theorem 2 we have to show that we do not create new conflict paths by inserting a bend at an edge e.

1. Let $P = e_1, \ldots, e_l$ be a propagating path between flanks or bowls, such that the number of right bends and the number of left bends along P should be equal. If a conflict path $P_c = e'_1, \ldots, e'_r$ enters P, runs along some edges together with P and finally leaves P, we have to ensure that we do not place the bend for P_c at one of the common edges. See Fig. 7a) for illustration. Let e be an edge lying on both paths. At least one of the endpoints of P_c, say e'_1 tries to force e in another direction as e_1 and e_l do. Thus there are also conflict paths P_{c_1} and P_{c_2} from e'_1 to e_1, and from e'_1 to e_l, respectively. To cut them we have to place either one bend *before* the common piece (the vertical cut in Fig. 7a)) or *two* bends in different directions along the path P (the two diagonal cuts in Fig. 7a)). In both cases the conditions for P are fulfilled.

2. Now we consider a path $P = e_1, \ldots, e_l$, where the difference between right and left bends has to be less or equal than 1. This situation occurs when exactly one node v between e_1 and e_l does not propagate directions while all the other do. Thus P consists of two parts each of them defining a propagating path. Therefore it is not possible that two conflict paths share disjoint sets of edges of one of these parts. Thus, to place two right bends on P, these bends have to be placed in different parts of P. Let $P_{c_1} = e_1, \ldots e_{l_1}$ and $P_{c_2} = e_{l_2}, \ldots, e_l$ be two conflict paths sharing some edges of P, such that the two common pieces lie at different sides of v. Since every node of P except v is propagating directions, $P'_c = e_{l_2}, \ldots, e_{l_1}$ is a conflict path, too, and we get a situation as in Fig. 7b). It is easy to see that every feasible cut fulfills the condition for P: If we cut P_{c_1} below v and P_{c_2} above v (the only way to place two right bends along P), we have to cut P'_c at its vertical part, yielding a left bend between e_1 and e_l; thus the difference between right and left bends is at most 1.

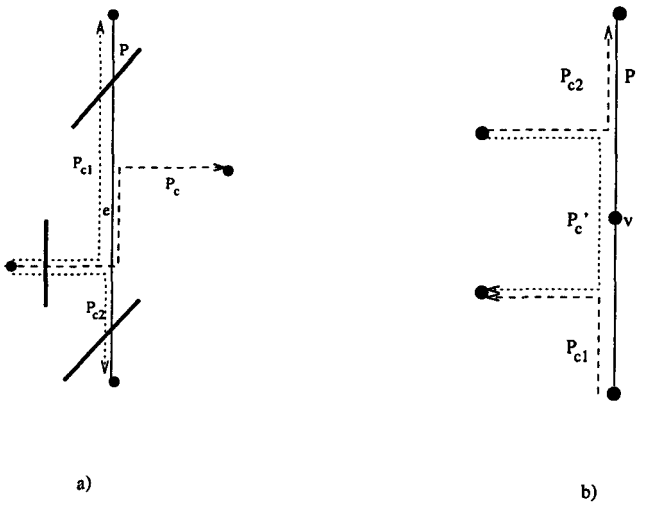

Fig. 7. Crossing paths

3. To prove the correctness of Algorithm 2 generally (the difference between the number of right and left bends along a path P has to be bounded by k) we only have to apply the argument of 2. to each connected component of P after removing the non-propagating nodes along P. This completes the proof of Theorem 2.

Theorem 3 *Algorithm 2 runs in* $O(n^2 \cdot \log n)$ *time.*

Proof. Let us first consider the expense for constructing the network: if we want to compute and write down all conflict paths the expense would be $\Theta(n^3)$ ($\Theta(n^2)$ conflict paths each having a length of $\Theta(n)$). But we do not need all this information to construct the network. It is sufficient to know for every edge e of the graph
– if e is part of a conflict path in $\hat{P}$ and if yes
– the predecessor of e in each of these conflict paths, if it exists, i.e. the edge being incident with e between e and the start point of the conflict path.
For each such edge e we establish three vertices v_e, v'_e, v''_e in the network and connect them with directed edges according to the information about the predecessor(s) of e.

How to get this information? For every edge e_d with determined direction (bowl or flank) we compute all propagating paths starting or ending at e_d and mark every edge e belonging to such a path with a subset of the direction set $\{l, v, r\}$ (due to the influence of e_d) and the predecessor of e in this propagating path, like in the beginning of this section. This procedure requires $O(n)$ time for every e_d, thus $O(n^2)$ in total. Now the edges are marked with no, one or several direction sets, and for every edge e being marked with more than one set, we check if there is a pair of disjoint direction sets. If yes, e is part of a conflict path and its neighbours in this path are its predecessors belonging to the corresponding direction subsets. Of course, computing the conflict paths of length one and deleting conflict paths which contain such paths of length one is easy. Thus we can construct the network in time $O(n^2)$.

The size of the network is linear: Since the number of nodes in the network is three times the number of edges in the input graph (plus some copies for paths of length one) and the input graph is planar, we only have a linear number of nodes in the network. The number of edges of the form (S, v), (v, T), (v, v') and (v', v'') is obviously linear, too, for the same reason. Edges of the form (v''_{e_1}, v_{e_2}) only appear, if e_1 and e_2 are consecutive edges in the graph, in particular they are incident to the same node. Therefore, for every node v''_{e_1} in the network there can only be three edges leaving it. So linearity of the number of edges of the network directly follows from the planarity of the graph. Thus both the number of nodes and edges in the network is linear and the min-cut-computation can be done in time $O(n^2 \cdot \log n)$ [15].

So the total algorithm runs in $O(n^2 \cdot \log n)$ time. ◇

5 The General Case

Now we construct layouts for general upward planar graphs without any restrictions. The idea is to transform an upward planar graph to a graph without structural bends; this can be done by running Tamassia's algorithm for the undirected graph and replacing every bend by an artificial node. The result is an upward planar graph, where we can apply our Algorithm 2. Let $G = (V, E)$ be an upward planar graph.

Algorithm 3

1. Compute G', the undirected version of G like in Section 4.
2. Run Tamassia's algorithm for G' and insert artificial nodes instead of the bends.
3. Assign a direction to every edge $\hat{e}$ in the resulting graph $\hat{G}'$: If $\hat{e}$ also exists in G it keeps its direction. Otherwise $\hat{e}$ is a part of an edge e in G and gets the same direction as e.
4. Run Algorithm 2 for this directed graph $\hat{G}$.
5. Replace every artificial node having an angle of 90° between its incident edges by a bend.

$$G \rightarrow_{(1)} G' \rightarrow_{(2)} \hat{G}' \rightarrow_{(3)} \hat{G} \rightarrow_{(4+5)} G^*$$

Theorem 4 *Algorithm 3 computes for an upward planar graph G and a planar embedding for it a region preserving OUPD G^* for G with at most $s + o$ bends, where s are the number of bends of G' and o are the number of bends of the optimal solution.*

Proof. Let G be an upward planar graph that requires o bends in its optimal orthogonal upward planar layout. Since every artificial node in $\hat{G}'$ has degree two, there is no new conflict path in $\hat{G}'$, i.e. every conflict path in $\hat{G}'$ also is a conflict path in G. Thus Algorithm 2 finds a layout for $\hat{G}'$ with at most o bends. Replacing s artificial nodes by bends can cause at most s further bends. ◇

Remark 1 *Since $o > s$, the number of bends in the output graph of Algorithm 3 exceeds the number of bends in the optimal layout at most by the factor 2.*

6 Open Problems

It would be interesting to have an efficient algorithm that guarantees optimality for general upward planar graphs. In this case the problem of finding a region preserving bend minimum OUPD of a planar graph would be solvable in polynomial time, whereas the corresponding problem of finding such a minimum over all planar topological embeddings is proved to be NP-hard. Another question of interest is to include additional criteria into consideration like the area or edge lengths needed by the drawings.

References

1. Bertolazzi, P., R.F. Cohen, G. Di Battista, R. Tamassia and I.G. Tollis, *How to Draw a Series-Parallel Digraph*, Proc. 3rd Scandinavian Workshop on Algorithm Theory, LNCS 621, (1992), pp. 272-283.
2. Di Battista, G., G. Liotta and F. Vargiu, *Spirality of Orthogonal Representations and Optimal Drawings of Series-Parallel Graphs and 3-planar Graphs*, Proc. 3nd Workshop on Algorithms and Data Structures, (1993), Lecture Notes in Comp. Science 709, pp. 151-162.
3. Di Battista, G., W.P. Liu, and I. Rival, *Bipartite Graphs, Upward Drawings and Planarity*, Information Processing Letters. vol. 36, (1990), pp. 317-322.
4. Di Battista, G. and R. Tamassia, *Algorithms for Plane Representations of Acyclic Digraphs*, Theoretical Computer Science Vol.61, (1988), pp. 175-198.
5. Di Battista, G., R. Tamassia and I.G. Tollis, *Area Requirement and Symmetry Display in Drawing Graphs*, Discrete and Comp. Geometry 7 (1992), pp. 381–401.
6. Eades, P., B. McKay and N. Wormald, *On an Edge Crossing Problem*, Proc. 9th Australian Computer Science Conf., (1986), pp. 327-334.
7. Fößmeier, U. and M. Kaufmann, *An Approach for Bend-Minimal Upward Drawing*, Workshop of GD'93, Paris (1993), pp. 27-29.
8. Garg, A. and R. Tamassia, *On the Computational Complexity of Upward and Rectilinear Planarity Testing*, Report CS-94-10, Comp. Sci. Dep., Brown Univ., Providence (1994), this proceedings.
9. Kelly, I. and I. Rival, *Planar Lattices*, Canadian J. Mathematics, Vol. 27, (1975), pp. 636-66.
10. Storer, J.A., *On Minimal Node-cost Planar Embeddings*, Networks 14 (1984), pp. 181–212.
11. Sugiyama, K, S. Tagawa and M. Toda, *Methods for Visual Understanding of Hierarchical Systems*, IEEE Trans. on Systems, Man and Cybernetics, Vol, SMC-11, (1981), pp. 109-125.
12. Tamassia, R., *On Embedding a Graph in the Grid with the Minimum Number of Bends*, SIAM J. Comput. 16 (1987), pp. 421-444.
13. Tamassia, R. and I.G. Tollis, *A Unified Approach to Visibility Representations of Planar Graphs*, Discrete and Comput. Geometry 1 (1986), pp. 321 - 341.
14. Tamassia, R. and I.G. Tollis, *Efficient Embedding of Planar Graphs in Linear Time*, Proc. IEEE Int. Symp. on Circuits and Systems, Philadelphia, (1987), pp. 495–498.
15. Tarjan, R.E., *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, (1983), Chapter 8.
16. Thomassen, C., *Planar Acyclic Oriented Graphs*, Order, Vol. 5, (1989), pp. 349-361.