Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 23, no. 3, pp. 499–524 (2019)
DOI: 10.7155/jgaa.00500

A Flow Formulation for Horizontal Coordinate Assignment with Prescribed Width

Michael Jünger[1] *Petra Mutzel*[2] *Christiane Spisla*[1]

[1]University of Cologne, Cologne, Germany
[2]TU Dortmund University, Dortmund, Germany

Abstract

We consider the coordinate assignment phase of the Sugiyama framework for drawing directed graphs in a hierarchical style. The extensive literature in this area has given comparatively little attention to a prescribed width of the drawing. We present a minimum cost flow formulation that supports prescribed width and optionally other criteria like lower and upper bounds on the distance of neighboring nodes in a layer or enforced vertical edge segments. In our experiments we demonstrate that our approach can compete with state-of-the-art algorithms.

Submitted:	Reviewed:	Revised:	Accepted:
November 2018	January 2019	June 2019	June 2019
	Final:	Published:	
	July 2019	September 2019	

Article type: Regular paper
Communicated by: T. Biedl and A. Kerren

E-mail addresses: mjuenger@informatik.uni-koeln.de (Michael Jünger) petra.mutzel@cs.tu-dortmund.de (Petra Mutzel) spisla@informatik.uni-koeln.de (Christiane Spisla)

1 Introduction

The Sugiyama framework [21] is a popular approach for drawing directed graphs. It layouts the graph in a hierarchical manner and works in five phases: Cycle removal, layer assignment, crossing minimization, coordinate assignment and edge routing. If the graph is not already acyclic, some edges are reversed to prepare the graph for the next phase. Then each node is assigned to a layer so that all edges point from top to bottom. After that the orderings of the nodes within each layer are determined. In the coordinate assignment phase that we consider here, the exact positions of the nodes are fixed. Finally the edges are drawn, e.g., as straight lines. A good overview over the different phases of the framework can be found in [15].

After the nodes are assigned to layers and the orderings of the nodes within their layers are fixed, the task of the coordinate assignment phase (and the focus of this paper) is to compute x-coordinates for all nodes. There are several, sometimes contradicting, objectives in this phase, e.g., short edges, minimum distance between neighboring nodes, straight edges, balanced positions of the nodes between their neighbors in adjacent layers, and few bend points of edges that cross multiple layers. The criterion "short edges" can be handled by exact algorithms as well as fast heuristics that give pleasant results, possibly also considering other aesthetic criteria.

When it comes to the width of the drawing one usually tries to restrict the maximum number of nodes in one layer during the layer assignment phase, see e.g. [8]. Long edges, i.e. edges that span more than two layers, are often split into paths with one dummy node on each intermediate layer. Healy and Nikolov [14] present a branch-and-cut approach to compute a layering that takes the influence of the number of dummy nodes on the width into account. Jabrayilov et al. [16] do the same in a mixed integer program that treats the first two phases of the Sugiyama framework simultaneously. Indeed, finding a layering with a bound on the width (and the height) considering original and dummy nodes is NP-hard [5]. Minimizing the height of a layering while complying with a given maximum width and considering only original nodes is NP-complete (via a reduction from *precedence-constrained multiprocessor scheduling problem*, see [13]), unless the maximum width is less than or equal to two. In this case, the Coffman-Graham algorithm [8] is exact. But still, the maximum number of nodes in one layer does not necessarily define the actual width of the final drawing, as illustrated in Figure 1. There we see that a minimum total horizontal edge length, i.e. the sum of the horizontal distances between the end points of all edges, and a minimum width can be two contradicting objectives. The main objective of most methods for the coordinate assignment phase is "short edges", which often leads to small drawings, but the width of the final layout is not directly addressed.

There may be further requirements for the final drawing, such as an aspect ratio in order to make optimal use of the drawing area [18], or a maximum distance between two nodes on the same layer if they are semantically related (compare the idea of compound graphs or cluster graphs, e.g. [11]). A common

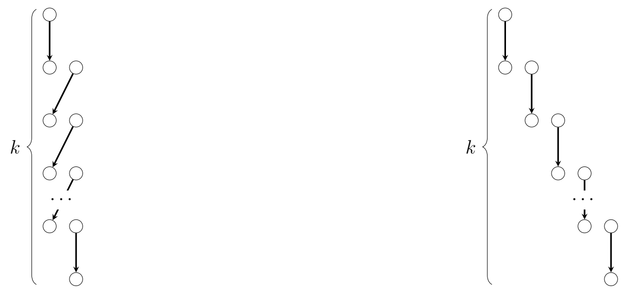


Figure 1: Two different drawings of a graph with $2(k-2)+2$ nodes and $k-1$ edges, where k is the number of layers the graph is drawn on. In the left picture the horizontal edge length is $k-3$ and the width is 1, in the right picture the horizontal edge length is 0 and the width is $k-2$.

request is that inner segments of long edges are drawn as vertical straight lines in order to improve readability.

Related work. Sugiyama et al. [21] present a quadratic programming formulation that has a combination of two asthetic criteria as objective function, short edges (closeness to adjacent nodes) and a balanced layout (positioning nodes close to the barycenter of their upper and lower neighbors). Gansner et al. [12] give a simpler formulation in which they replace quadratic terms of the form $(x_v - x_u)^2$ by $|x_v - x_u|$ and leave out the balance terms. The coordinate assignment problem can be interpreted as an instance of the layer assignment problem, and they suggest to apply the network simplex algorithm to an auxiliary graph to obtain a drawing with minimum horizontal edge length. Given an initial layout, some heuristics sweep through the layers and try to shift the nodes to better positions depending on the fixed x-coordinates of their neighbors in adjacent layers, see e.g. [10, 20, 21]. Two fast heuristics that compute coordinates given only the layering and the ordering, but no initial layout, are presented by Buchheim et al. [6] and by Brandes and Köpf [4]. Both algorithms draw inner segments of long edges straight and aim for a balanced layout with short edges.

Our Contribution. We formulate the coordinate assignment problem as a minimum cost flow problem that can be solved efficiently. Within this formulation we can fix the maximum width of the final drawing as well as a maximum and minimum horizontal distance between nodes in the same layer and we can enforce straightness of some edges. We compute x-coordinates such that the total horizontal edge length is minimized subject to these further constraints. In particular, we can compute a hierarchical layout with minimum total horizontal edge length (Theorem 1) and width-minimum layouts with shortest possible horizontal edge length (Theorem 2). In Section 4, we compare our approach to state-of-the-art algorithms in terms of the drawing width, the total horizontal edge length and the computation time.

2 Notation and Preliminaries

Let $G = (V, E)$ be a directed graph with $|V| = n$ nodes and $|E| = m$ edges. For a directed edge $e = (u, v)$, $\mathit{start}(e) = u$ denotes the start node of e and $\mathit{target}(e) = v$ denotes the target node of e. A *path* P from u to v of length k is a set of edges $\{e_i = (v_i, v_{i+1}) \mid i = 1, \dots, k$ where $u = v_1$, $v = v_{k+1}$ and $v_i \neq v_j$ for $1 \leq i, j \leq k\}$. We also write $u \overset{*}{\to} v$ for such a path. If $v_{k+1} = v_1$ it is called a *cycle*. A graph is called a *directed acyclic graph (DAG)* if it has no cycles. A *layering* $\mathcal{L}$ of a graph assigns every $v \in V$ to a *layer* L_i, such that $i < j$ holds for every edge $e = (u, v)$ with $\mathcal{L}(u) = L_i$ and $\mathcal{L}(v) = L_j$. The layering is called *proper* if $\mathcal{L}(v) = \mathcal{L}(u) + 1$ for every edge (u, v), i.e., the layers of every pair of adjacent nodes are consecutive. An edge that violates the latter property is called a *long edge*. Every graph with a layering can be transformed into a graph with a proper layering by subdividing every long edge into a chain of edges. We use $|\mathcal{L}|$ to denote the number of layers and $|L_i|$ to denote the number of nodes in layer L_i.

An *ordering ord* defines a permutation of the nodes of each layer. For every layer L_i it assigns each node in L_i its position within the layer. We write $u < v$ if $\mathit{ord}(u) < \mathit{ord}(v)$ and u and v are in the same layer. The ordering *ord* defines a partial ordering on V. We use v_j^i to denote the j-th node in layer L_i.

Given a graph G with a layering $\mathcal{L}$ and an ordering *ord*, the *horizontal coordinate assignment problem (HCAP)* asks for x-coordinates for every node so that $x(u) < x(v)$ if $u < v$. We will restrict ourselves to integer coordinates. The *horizontal length* of an edge $e = (u, v)$ is defined as $\mathit{length}(e) = |x(v) - x(u)|$ and the *total horizontal edge length* is $\mathit{length}(E) = \sum_{e \in E} \mathit{length}(e)$. The *width* of the assignment is $\max_{v \in V} x(v) - \min_{v \in V} x(v)$. Unless otherwise stated, we mean the horizontal length whenever we talk about the length of an edge.

$\mathit{HCAP}_{\mathit{minEL}}$ is the variant of *HCAP* in which we also want to mimimize the total horizontal edge length and $\mathit{HCAP}_{\mathit{minW}}$ asks for x-coordinates that minimize the width of the assignment. We use $\mathit{HCAP}_{\mathit{minW\text{-}EL}}$ to denote the problem of finding a horizontal coordinate assignment with minimum width and that has the shortest total horizontal edge length among all width-minimum coordinate assignments.

We assume familiarity with minimum cost flows. Ahuja et al. [1] give a good overview. Let $N = (V_N, E_N)$ be a directed graph with a single source s and a single sink t, so for all other nodes the amount of incoming flow equals the amount of outgoing flow. We have lower and upper bounds on the edges and a cost function $\mathit{cost} : E_N \to \mathbb{R}$. Let f be a feasible flow. For a subset of nodes $V' \subseteq V_N \setminus \{s, t\}$ we use $f(V') = \sum_{v \in V'} \sum_{e=(v,w)} f(e) = \sum_{v \in V'} \sum_{e=(u,v)} f(e)$ to denote the *flow through* V'. For s we define $f(s)$ to be the total amount of flow leaving s. For a subset of edges $E' \subset E_N$ we use $f(E') = \sum_{e \in E'} f(e)$ to denote the flow over E' and with $\mathit{cost}(E') = \sum_{e \in E'} \mathit{cost}(e)$ the cost of E' and with $\mathit{cost}_f = \sum_{e \in E_N} f(e) \cdot \mathit{cost}(e)$ the total cost of f.

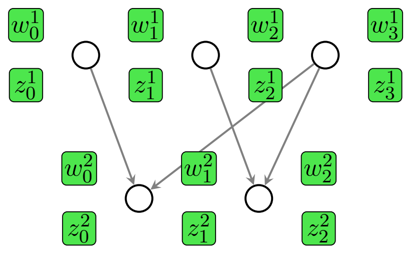


Figure 2: Example graph together with the network nodes. Nodes of G are white circles, nodes of N are green rectangles. Edges of G are gray.

3 Network Flow Formulation

In this section we describe the construction of a network for the horizontal coordinate assignment problem on proper layered instances. Given a minimum cost flow in this network we show how to obtain x-coordinates for all nodes such that the total horizontal edge length is minimized. Minimizing edge lengths with flow techniques goes back to the compaction of orthogonal layouts, see e.g. [9]. By a simple modification we can compute x-coordinates that give us minimum total horizontal edge length with respect to a given maximum width of the drawing. The basic idea is that flow represents horizontal distance and we send flow from top to bottom through the layers.

3.1 Network Construction

Let $G = (V, E)$ be a DAG with a proper layering $\mathcal{L}$ and an ordering *ord*. We construct a flow network $N = (V_N, E_N)$ to model the coordinate assignment problem. For now we assume that consecutive nodes within a layer must be placed so that they are at least a distance of one apart and that we have no further requirements concerning the edges.

For every layer L_i with $i \in \{1, \dots, |\mathcal{L}|\}$ we add nodes $w_0^i, w_1^i, \dots, w_{|L_i|}^i$ and $z_0^i, z_1^i, \dots, z_{|L_i|}^i$ to N. Imagine the node w_j^i placed above the layer L_i and between v_j^i and v_{j+1}^i (w_0^i is placed at the left end and $w_{|L_i|}^i$ at the right end of the layer), see Figure 2. The nodes z_j^i are placed in the same way below layer L_i. Although we do not have a drawing of G at this moment we can still use terms like "above" and "below" because the layering gives us a vertical ordering of the nodes of G and we can talk about "left" and "right" because of the given ordering of the nodes in each layer. Since we are placing the nodes w_j^i and z_j^i "between" the nodes v_j^i and v_{j+1}^i we extend the "<" relation to give a partial ordering on $V \cup V_N$ in the following way: $w_0^i < v_1^i < w_1^i < v_2^i < \cdots < v_{|L_i|}^i < w_{|L_i|}^i$ and $z_0^i < v_1^i < z_1^i < v_2^i < \cdots < v_{|L_i|}^i < z_{|L_i|}^i$. We connect w_j^i to z_j^i with an edge a_j^i that has a lower bound of one and an upper bound of ∞ and a cost of zero. The flow over these edges will define the distance between v_j^i and v_{j+1}^i. Let A denote the set of these edges.

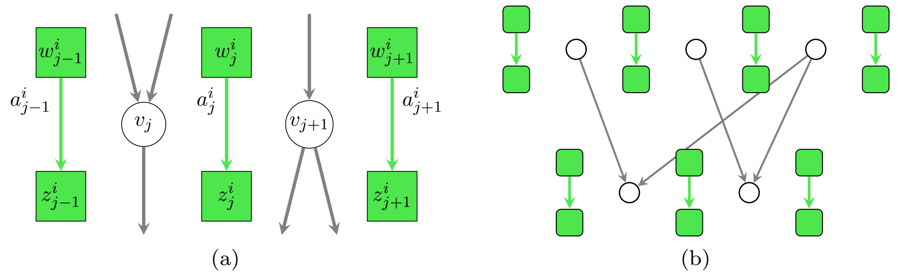
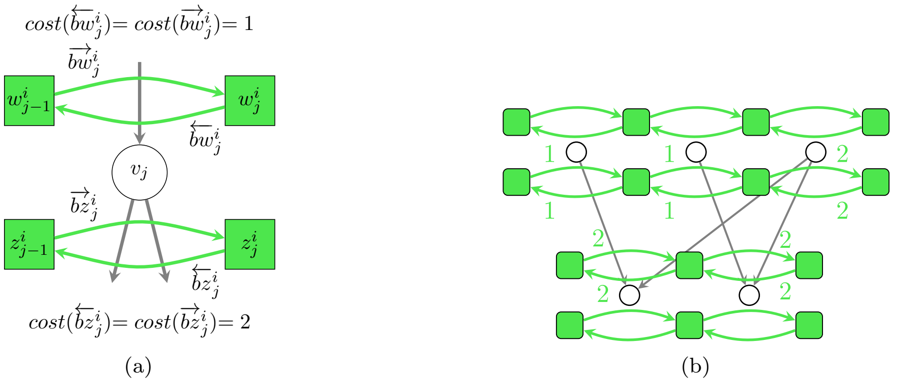


Figure 3: (a) Illustration of edges of the set A. (b) Example graph together with the edges of the set A. Nodes of G are white circles, nodes of N are green rectangles. Edges of G are gray, edges of N are green. All depicted edges have no cost.



$cost(\overleftarrow{bw}^i_j) = cost(\overrightarrow{bw}^i_j) = 1$
$\overrightarrow{bw}^i_j$
w^i_{j-1}
w^i_j
$\overleftarrow{bw}^i_j$
v_j
$\overrightarrow{bz}^i_j$
z^i_{j-1}
z^i_j
$\overleftarrow{bz}^i_j$
$cost(\overleftarrow{bz}^i_j) = cost(\overrightarrow{bz}^i_j) = 2$
(a)
1
1
2
1
1
2
2
2
2
2
(b)

Figure 4: (a) Illustration of edges of the set B. (b) Example graph together with the edges of the set B. Nodes of G are white circles, nodes of N are green rectangles. Edges of G are gray, edges of N are green. Labels on network edges denote the cost, unlabeled edges have no cost.

For every layer L_i with $i \in \{1, \ldots, |\mathcal{L}|\}$ and every $j \in \{1, \ldots, |L_i|\}$ we add edges $\overrightarrow{bw}^i_j = (w^i_{j-1}, w^i_j)$, $\overleftarrow{bw}^i_j = (w^i_j, w^i_{j-1})$, $\overrightarrow{bz}^i_j = (z^i_{j-1}, z^i_j)$ and $\overleftarrow{bz}^i_j = (z^i_j, z^i_{j-1})$ to N, see Figure 4. The lower bound of these edges is zero and the upper bound is ∞. The cost of these network edges equals the number of graph edges they "cross over". That means the cost of $\overleftarrow{bw}^i_j$ and $\overrightarrow{bw}^i_j$ equals the number of incoming graph edges of node v^i_j and the cost of $\overleftarrow{bz}^i_j$ and $\overrightarrow{bz}^i_j$ equals the number of outgoing graph edges of v^i_j. Positive flow over one of these edges will cause the crossed-over graph edges to have positive horizontal length. Because of the positive edge cost, a minimum cost flow will never use $\overrightarrow{bw}^i_j$ and $\overleftarrow{bw}^i_j$ simultaneously (the same holds for $\overrightarrow{bz}^i_j$ and $\overleftarrow{bz}^i_j$). Let B denote the set of these edges.

Now we connect the nodes of neighboring layers. We could add edges be-

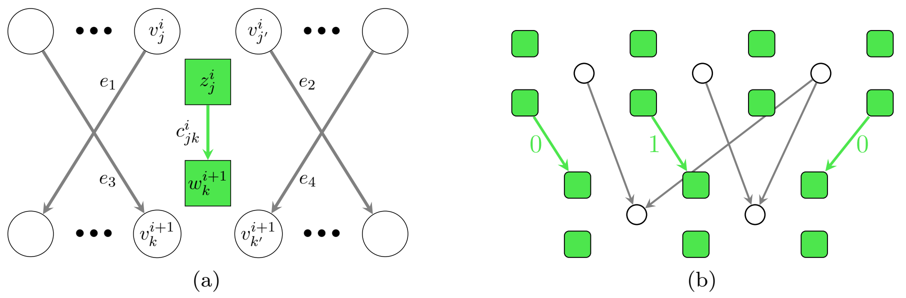


Figure 5: (a) Illustration of edges of the set C. (b) Example graph together with the edges of the set C. Nodes of G are white circles, nodes of N are green rectangles. Edges of G are gray, edges of N are green. Labels on network edges denote the cost.

tween every z_j^i and every w_k^{i+1}, but we want to keep the number of edges between layers as small as possible. We add edges only in special situations and will show later that this suffices for correctness. For every layer L_i with $i \in \{1, \ldots, |\mathcal{L}|-1\}$ we add edges $c_{00}^i = (z_0^i, w_0^{i+1})$ and $c_{|L_i||L_{i+1}|}^i = (z_{|L_i|}^i, w_{|L_{i+1}|}^{i+1})$ to the network with a lower bound of zero, an upper bound of ∞ and a cost of zero. Additionally (see Figure 5(a) for an illustration) we add edges $c_{jk}^i = (z_j^i, w_k^{i+1})$ if there exist $e_1, e_2, e_3, e_4 \in E$ with $start(e_1) = v_j^i$, $start(e_2) = v_{j'}^i$, where $v_{j'}^i$ is the next node to the right of v_j^i with an outgoing edge and $target(e_3) = v_k^{i+1}$, $target(e_4) = v_{k'}^{i+1}$, where $v_{k'}^{i+1}$ is the next node to the right of v_k^{i+1} with an incoming edge and the following conditions hold: $start(e_3) \leq start(e_1) < start(e_2) \leq start(e_4)$ and $target(e_1) \leq target(e_3) < target(e_4) \leq target(e_2)$. We call this situation a *hug* between z_j^i and w_k^{i+1}. These edges get a lower bound of zero, an upper bound of ∞, and the cost equals the number of graph edges they cross over: $cost(c_{jk}^i) = |\{e = (v_p^i, v_q^{i+1}) \in E \mid p \leq j \wedge q \geq k' \text{ or } p \geq j' \wedge q \leq k\}|$. Like the edges of B, flow on edges of this kind will cause horizontal length and we use C to denote the set of all c_{jk}^i. Figure 5(b) shows all edges of the set C for one example graph.

Finally we add a source s and a sink t to the network. We connect s with every w_j^1, $j \in \{0, \ldots, |L_1|\}$ and t with every $z_k^{|\mathcal{L}|}$, $k \in \{0, \ldots, |L_{|\mathcal{L}|}|\}$. These edges get a lower bound of zero, an upper bound of ∞ and a cost of zero. Figure 6(a) shows a complete example network. In Figure 6(b) a feasible flow in the example network is depicted. In the next subsection we show how a drawing is derived from a feasible flow.

In this network flow can only run "from top to bottom ". Although flow can be shifted among the w and z nodes within each layer (by arcs of the set B), it eventually has to take one of the arcs of the set C or A, respectively, to proceed to t. Once a unit of flow has passed "through" a layer, it cannot go back. If it is clear from the context which layer or which node is meant, we omit the nodes' subscripts and superscripts.

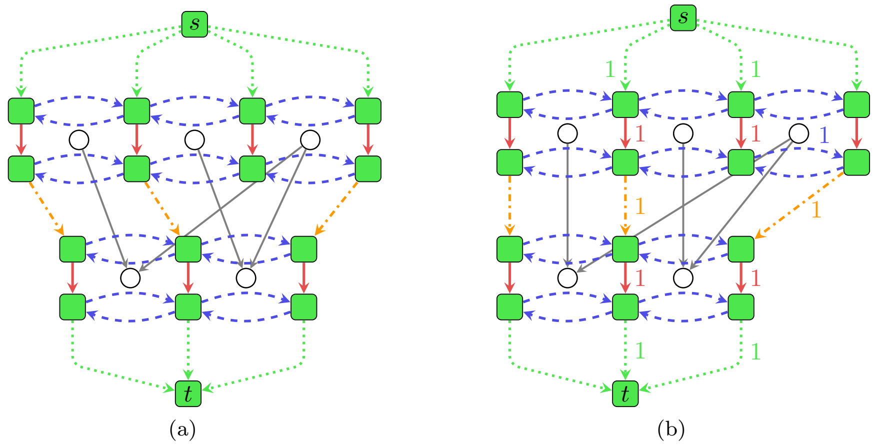


Figure 6: (a) Example graph together with the full network. Nodes of G are white circles, nodes of N are green rectangles. Edges of the set A are red (solid), edges of the set B are blue (dashed) and edges of the set C are orange (dashdotted). Network edge costs are omitted. (b) Network with feasible flow shown as labels on network edges. Unlabeled edges carry no flow. The induced drawing is also shown.

Lemma 1 *The network described above has* $\mathcal{O}(|V|)$ *nodes and* $\mathcal{O}(\min\{|V|^2, |V| + |E| + cross\})$ *edges, where cross is the number of crossings in* G *with the layering* $\mathcal{L}$ *and the ordering ord.*

Proof: Basically we have two network nodes for each graph vertex. For v_j^i we have one w_j^i and one z_j^i plus one w_0^i and one z_0^i for each layer. In addition we have the source s and the sink t. That gives us $2|V| + 2|\mathcal{L}| + 2 = \mathcal{O}(|V|)$ network nodes.

Now for the network edges. The set A contains $|V| + |\mathcal{L}|$ network arcs, one to the right of each graph node and one to the left of each layer. The set B consists of two arcs above and below each graph node. Therefore we have $|B| = 4|V|$.

The set C is the crucial one. In the worst case the nodes z^i and w^{i+1} build a complete bipartite subgraph for every i. For instance if every pair of consecutive layers of the input graph gives rise to a complete bipartite subgraph, then for every z_j^i and w_k^{i+1} we have a hug and therefore an arc. This would result in $\mathcal{O}(|V|^2)$ edges in the set C. But the number of hug situations, and therefore the size of C, is influenced by the number of edges of G and the way they cross. Every edge e could be the right side of a hug , i.e. $e = e_1 = e_3$. Similarly, every pair of crossing edges $e' = (v_{j_1}^i, v_{k_1}^{i+1})$ and $e'' = (v_{j_2}^i, v_{k_2}^{i+1})$ with $v_{j_2}^i < v_{j_1}^i$ and $v_{k_1}^{i+1} < v_{k_2}^{i+1}$ could lead to a hug between $z_{j_1}^i$ and $w_{k_2}^{i+1}$ (i.e. $e' = e_1$ and $e'' = e_3$). So the number of edges of G plus the number of crossings in G is another upper bound for the size of C.

□

$k \backslash j$	0	1	2	3
0	0	1	2	4
1	0	0	1	2
2	0	0	0	0

(a)

$k \backslash j$	1	2	3
0	1	1	2
1	0	1	1
2	0	0	0

(b)

Table 1: The values for (a) $\overrightarrow{p}^{1}_{jk} = |\{e = (v^1_\ell, v^2_r) \in E \mid v^1_\ell < z^1_j \text{ and } v^2_r > w^2_k\}|$ and (b) $\overrightarrow{q}^{1}_{jk} = |\{e = (v^1_j, v^2_r) \in E \mid v^2_r > w^2_k)\}|$ for the example graph in Figure 6.

Lemma 2 *The network described above can be constructed in $\mathcal{O}(|V|^2)$ time.*

Proof: The nodes of the network together with the edges of the sets A and B can be built in $\mathcal{O}(|V|)$ time.

For the construction of the arc set C all $\mathcal{O}(|V^2|)$ possibilities for arcs need to be checked. Checking if we have a hug between z^i_j and w^{i+1}_k and inserting the appropriate arc into the network can be achieved in constant time with the right data structure. Suppose we have direct access to the leftmost and rightmost, respectively, incoming and outgoing edges for every node $v \in V$. Then we check if the target node of the leftmost outgoing edge of v^i_j is to the left of w^{i+1}_k. This edge is a candidate for e_1 from the hug definition. If the target node of the rightmost outgoing edge of v^i_{j+1}(or the next node to the right, if v^i_{j+1} has no outgoing edges) is to the right of w^{i+1}_k, we have found a suitable edge for e_2. Similarly, we identify appropriate e_3 and e_4, if they exist, and add a network arc $c^i_{jk} = (z^i_j, w^{i+1}_k)$, if we have a hug situation.

To determine the cost of c^i_{jk} we use dynamic programming. The set of graph edges that are crossed by c^i_{jk}, which defines its arc cost, can be divided into the set of edges that start to the left of z^i_j and end to the right of w^{i+1}_k and the set of edges that start to the right of z^i_j and end to the left of w^{i+1}_k. Let us focus on the number of edges from the left of z^i_j to the right of w^{i+1}_k and let $\overrightarrow{p}^{i}_{jk}$ denote this number, see Table 1(a). The number of edges from right to left can be computed similarly. We can recursively compute $\overrightarrow{p}^{i}_{jk}$ for $0 \le j \le |L_i|$ and $1 \le k \le |L_{i+1}|$, regardless of if there exists an arc c^i_{jk} or not, exploiting the fact that $\overrightarrow{p}^{i}_{0k} = 0$ for all k and $\overrightarrow{p}^{i}_{jk} = \overrightarrow{p}^{i}_{j-1,k} + \overrightarrow{q}^{i}_{jk}$, where $\overrightarrow{q}^{i}_{jk} = |\{e = (v^i_j, v^{i+1}_r) \in E \mid v^{i+1}_r > w^{i+1}_k)\}|$ is the number of edges that start at v^i_j (directly to the left of z^i_j) and end anywhere to the right of w^{i+1}_k, see Table 1(b). This number can be precomputed for all j and k in $\mathcal{O}(|V^2|)$ time in total: We start with $\overrightarrow{q}^{i}_{j|L_i+1|} = 0$ for all $1 \le j \le |L_i|$ and set $\overrightarrow{q}^{i}_{jk} = \overrightarrow{q}^{i}_{j,k+1} + 1$ if there exists an edge (v^i_j, v^{i+1}_{k+1}) or $\overrightarrow{q}^{i}_{jk} = \overrightarrow{q}^{i}_{j,k+1}$ otherwise (assuming G is simple). So it takes $\mathcal{O}(|V^2|)$ time to construct the flow network.

□

3.2 Obtaining Coordinates and Correctness

Let f be a feasible flow in the network described above. We define the x-coordinate of a node v^i_j as

$$x(v^i_j) := \sum_{l=0}^{j-1} f(a^i_\ell). \tag{1}$$

Together with $y(v^i_j) = i$ we get an induced drawing with a feasible coordinate assignment, because for every v_j, v_k within the same layer $x(v_j) < x(v_k)$ if and only if $v_j < v_k$ (since the amount of flow over edges a $\in A$ is always positive).

Now we want to explain the correspondence between the cost of a flow f and the total horizontal edge length of the resulting drawing. The intuition is, that if flow is sent from the right of *start*(e) to the left of *target*(e) for some edge e, then *target*(e) is "pushed" to the right because of the additional flow on the left. This results in a horizontal expansion of e. We define for an edge $e = (u, v) \in E$

$$\begin{aligned}\overrightarrow{E}(e) \;\; := \;\; & \{bw \in B \mid \mathit{start}(bw) < v \;\wedge\; \mathit{target}(bw) > v\} \\ & \cup \{bz \in B \mid \mathit{start}(bz) < u \;\wedge\; \mathit{target}(bz) > u\} \\ & \cup \{c \in C \mid \mathit{start}(c) < u \;\wedge\; \mathit{target}(c) > v\}\end{aligned}$$

as the set of network edges that start to the left of e and end to the right of e, thus cross over e from left to right. Analogously the set of network edges that cross over a graph edge from right to left is

$$\begin{aligned}\overleftarrow{E}(e) \;\; := \;\; & \{bw \in B \mid \mathit{start}(bw) > v \;\wedge\; \mathit{target}(bw) < v\} \\ & \cup \{bz \in B \mid \mathit{start}(bz) > u \;\wedge\; \mathit{target}(bz) < u\} \\ & \cup \{c \in C \mid \mathit{start}(c) > u \;\wedge\; \mathit{target}(c) < v\}.\end{aligned}$$

We make the following observations:

Property 1 *For every $g \in B \cup C$ we have:*
$\mathit{cost}(g) = |\{e \in E \mid g \in \overrightarrow{E}(e)\}| + |\{e \in E \mid g \in \overleftarrow{E}(e)\}|$.

Property 2 *The amount of flow that passes through a layer is the same for every layer: $\sum_{j=0}^{|L_i|} f(a^i_j) = \sum_{j=0}^{|L_k|} f(a^k_j) = f(s)$ for all $i, k \in \{1, \ldots, |\mathcal{L}|\}$.*

Property 3 *The width of the induced drawing is*
$\max_{1 \le i \le |\mathcal{L}|} \left(\sum_{j=1}^{|L_i|-1} f(a^i_j) \right) \le f(s)$.

Property 4 *Let $e = (v^i_j, v^{i+1}_k)$ be an edge. Then*
$\sum_{\mathit{start}(a^{i+1}_\ell) < v^{i+1}_k} f(a^{i+1}_\ell) = \sum_{\mathit{target}(a^i_\ell) < v^i_j} f(a^i_\ell) + f(\overleftarrow{E}(e)) - f(\overrightarrow{E}(e))$.

The last property is illustrated in Figure 7. The total flow over all a^{i+1}_ℓ that are to left of *target*(e) can be divided into flow $f(\overleftarrow{E}(e))$ that crosses e from right to left and flow that does not cross e. The latter has to pass exclusively through nodes to the left of e and comes over some a^i_ℓ to the left of $\mathit{start}(e)$ $(\sum_{\mathit{target}(a^i_\ell)<\mathit{start}(e)} f(a^i_\ell))$. Flow over an arc a^i_ℓ to the left of $\mathit{start}(e)$ that does not end up over some a^{i+1}_ℓ to the left of $\mathit{target}(e)$ has to cross e from left to right $(f(\overrightarrow{E}(e)))$.

Lemma 3 *For a feasible flow f and its induced drawing, $\mathit{cost}_f \geq \mathit{length}(E)$ holds.*

Proof: Let $e = (v^i_j, v^{i+1}_k)$ be an edge of G. The length of e is $\mathit{length}(e) = |x(v^i_j) - x(v^{i+1}_k)|$ and together with (1) we have

$$\begin{aligned}
\mathit{length}(e) &= \Big|\sum_{l=0}^{j-1} f(a^i_\ell) - \sum_{l=0}^{k-1} f(a^{i+1}_\ell)\Big| \\
&= \Big|\sum_{\mathit{target}(a^i_\ell)<v^i_j} f(a^i_\ell) - \sum_{\mathit{start}(a^{i+1}_\ell)<v^{i+1}_k} f(a^{i+1}_\ell)\Big| \\
&= \Big|f(\overrightarrow{E}(e)) - f(\overleftarrow{E}(e))\Big| \qquad \text{(by Property 4).}
\end{aligned}$$

For the total edge length we obtain:

$$\begin{aligned}
\mathit{length}(E) &= \sum_{e\in E}\Big|f(\overrightarrow{E}(e)) - f(\overleftarrow{E}(e))\Big| \\
&\leq \sum_{e\in E}\Big(\Big|f(\overrightarrow{E}(e))\Big| + \Big|f(\overleftarrow{E}(e)\Big|\Big) \\
&= \sum_{e\in E}\Big(f(\overrightarrow{E}(e)) + f(\overleftarrow{E}(e)\Big) \\
&= \sum_{g\in E_N} f(g)\cdot|\{e\in E \mid g\in \overrightarrow{E}(e)\}\cup\{e\in E \mid g\in \overleftarrow{E}(e)\}| \\
&= \sum_{g\in E_N} f(g)\cdot \mathit{cost}(g) \qquad \text{(by Property 1)} \\
&= \mathit{cost}_f.
\end{aligned}$$

□

Lemma 4 *Let Γ be a drawing of G. There exists a flow f that induces Γ and whose cost is equal to the total edge length of Γ.*

Proof: If necessary, we set $x(v) := x(v) - \min_{v\in V} x(v)$ so that the smallest x-coordinate is zero. That gives us an equivalent drawing. Thus, there is always a feasible flow that leads to the drawing Γ. We will argue that there is such a feasible flow whose cost is equal to the total edge length of the drawing.

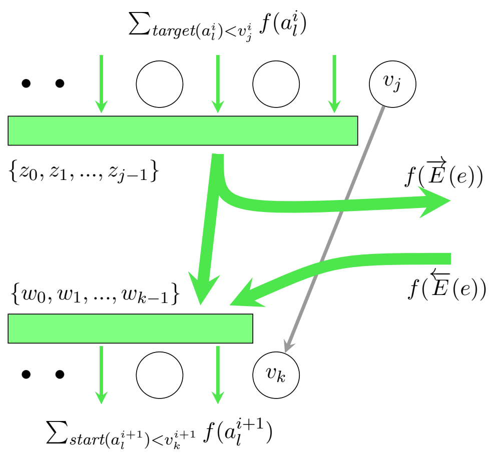

$\sum_{target(a_l^i)<v_j^i} f(a_l^i)$
v_j
$\{z_0, z_1, ..., z_{j-1}\}$
$f(\overrightarrow{E}(e))$
$f(\overleftarrow{E}(e))$
$\{w_0, w_1, ..., w_{k-1}\}$
v_k
$\sum_{start(a_l^{i+1})<v_k^{i+1}} f(a_l^{i+1})$

Figure 7: Illustration of Property 4. The rectangles represent all network nodes to the left of v_j and v_k, respectively. The thick arrows represent the flow over several network edges.

Let ω be the width of Γ. We send ω units of flow from s to t, so that the k-th unit takes the path $P_k = s \xrightarrow{*} w^1_{j_1} \xrightarrow{*} w^2_{j_2} \xrightarrow{*} \cdots \xrightarrow{*} w^{|\mathcal{L}|}_{j_{|\mathcal{L}|}} \xrightarrow{*} t$, where $w^i_{j_i}$ is chosen so that $x(v^i_{j_i+1}) \geq k$ and $x(v^i_{j_i}) < k$ ($w^i_{j_i} = w^i_0$, if $x(v^i_1) \geq k$ and $w^i_{j_i} = w^i_{|L_i|}$, if $x(v^i_{|L_i|}) < k$). That means we send the k-th unit through the k-th "column" of Γ. This is always possible, because we always have the subpaths $w^i_{j_i} \to z^i_{j_i} \xrightarrow{*} z^i_0 \to w^{i+1}_0 \xrightarrow{*} w^{i+1}_{j_{i+1}}$. So for every v there are $x(v)$ units of flow that pass by to the left of v, thus giving us correct coordinates for all nodes.

We define $E^i_k := \{e = (v^i_j, v^{i+1}_\ell) \in E \mid x(v^i_j) < k \text{ and } x(v^{i+1}_\ell) \geq k\} \cup \{e \in E \mid x(v^i_j) \geq k \text{ and } x(v^{i+1}_\ell) < k\}$, i.e. all edges that cross over the k-th column between L_i and L_{i+1}. We show that there exists a path P_k that produces the same cost as the number of graph edges that cross over the k-th column in total, that is $cost(P_k) = \sum_{i=1}^{|\mathcal{L}|-1} |E^i_k|$. Then we have $\sum_{k=1}^{\omega} cost(P_k) = \sum_{k=1}^{\omega} \sum_{i=1}^{|\mathcal{L}|-1} |E^i_k| = length(E)$ and we have proven the lemma.

Since we have a proper layered drawing, it suffices to focus on the subpath P^i_k from $z = z^i_{j_i}$ to $w = w^{i+1}_{j_{i+1}}$ between two consecutive layers. Notice that network edges (s, w^1), (w^i, z^i) and $(z^{|\mathcal{L}|}, t)$ do not contribute to the cost of the flow. For better readability we use u_j to denote the nodes of L_i and v_j for the nodes of L_{i+1} and we omit the superscripts. If not stated otherwise, we use z_j for z^i_j and w_j for w^{i+1}_j. We construct $P' = P^i_k$ so that $cost(P') = |E'| = |E^i_k|$.

Case 1: There is no edge e with $start(e) < z$ and $target(e) < w$.
That means every edge e with $start(e) < z$ has $target(e) > w$, and if $target(e) < w$ then $start(e) > z$. Then we set $P' = z \to z_{j_i - 1} \xrightarrow{*} z_0 \to w_0 \to w_1 \xrightarrow{*} w$. For every $u_j < z$ with p outgoing edges, P' uses exactly one $\overleftarrow{bz}$ with cost p. All these edges are in E'. For every $v_j < w$ with q incoming edges we use exactly

one $\overrightarrow{bw}$ with cost q. Again these edges are in E'. So $cost(P') = |E'|$, since there are no other edges in E'.

Case 2: There is no edge e with $start(e) > z$ and $target(e) > w$.
Symmetrically to Case 1, we set $P' = z \xrightarrow{*} z_{|L_i|} \rightarrow w_{|L_{i+1}|} \xrightarrow{*} w$. As before the cost of P' equals $|E'|$.

Case 3: There is an edge e_ℓ with $start(e_\ell) < z$ and $target(e_\ell) < w$ and another edge e_r with $start(e_r) > z$ and $target(e_r) > w$.
Let e_ℓ be the edge with the biggest $x(start(e))$ of all edges e with $start(e) < z$ and $target(e) < w$, and let e_r be the edge with the smallest $x(start(e))$ of all edges e with $start(e) > z$ and $target(e) > w$.

Case 3.1: There is at least one node u' with outgoing edges and $start(e_\ell) < u' < z$.
The situation is depicted in Figure 8 and described as follows. Let $u_g = start(e_\ell)$ and $v_{g'} = target(e_\ell)$. We know $v_{g'} < w$. Let $v_{h'}$ be the first node to the right of $v_{g'}$ with an edge $e_{r'} = (u_h, v_{h'})$ and $u_h > u_g$. Such a node exists, since we have e_r. Notice that $v_{h'}$ might be to the right of w.

Then we have a hug: Set $e_1 = e_\ell$, set e_2 to one outgoing edge of u_{g+1} (or the next node to the right of u_g, which has an outgoing edge), $e_4 = e_{r'}$ and set e_3 to one incoming edge of $v_{h'-1}$ (or the next one to the left of $v_{h'}$), see Figure 8. Notice that e_1 may coincide with e_3 and e_2 with e_4.

We have $start(e_3) \leq start(e_1)$, because we chose $e_1 = e_\ell$ with the biggest $x(start(e))$ and $v_{h'}$ is the first node to the right of $v_{g'}$ with an adjacent node to the right of u_g. So every node between $v_{g'} = target(e_1)$ and $v_{h'}$, including $v_{h'-1} = target(e_3)$, can only have adjacent nodes to the left of $u_g = start(e_1)$. It is clear that $start(e_1) < start(e_2)$ and $start(e_2) \leq start(e_4)$, since $start(e_4) = u_h > u_g$. By the choice of e_1, e_3 and e_4, $target(e_1) \leq target(e_3) < target(e_4)$ holds. We know that $target(e_2) > w$ because there is at least one node between $start(e_1)$ and z whose outgoing edges have to end to the right of w because of the choice of e_1. If $target(e_4) > target(e_2)$ then e_2 would have been chosen for $e_{r'}$ and therefore for e_4. So $target(e_4) \leq target(e_2)$ also holds. So there exists $c_{g(h'-1)} \in E_N$ and we set $P' = z \xrightarrow{*} z_g \rightarrow w_{h'-1} \xrightarrow{*} w$.

Now for the cost. A subset of E' are the edges e with $z_g < start(e) < z$ and $target(e) > w$, which are covered by the $\overleftarrow{bz}$ of P'.

We have two options. First, if $w_{h-1} > w$ then all edges e with $start(e) < z_g$ and $target(e) > w_{h'-1}$ are crossed by $c_{g(h'-1)}$ and therefore their lengths are taken into account. The lengths of the remaining edges e with $start(e) < z_g$ and $w < target(e) < w_{h'-1}$ are accounted for by the $\overleftarrow{bw}$ of P'. Edges e with $start(e) > z > u_g$ and $target(e) < w < w_{h'}$ are also considered by $c_{g(h'-1)}$. There cannot be any edge e with $z < start(e)$ and $w < target(e) < w_{h'-1}$ or $z_g < start(e) < z$ and $target(e) < w$, which would be crossed over by two different edges of P', due to the choice of edges e_1 to e_4.

Second, if $w_{h-1} < w$ then $c_{g(h'-1)}$ covers all edges e with $start(e) < z_g$ and $target(e) > w > w_{h'-1}$ and all edges e with $start(e) > z > z_g$ and $target(e) < w_{h'-1} < w$. Edges e with $start(e) > z$ and $w_{h'-1} < target(e) < w$ are covered

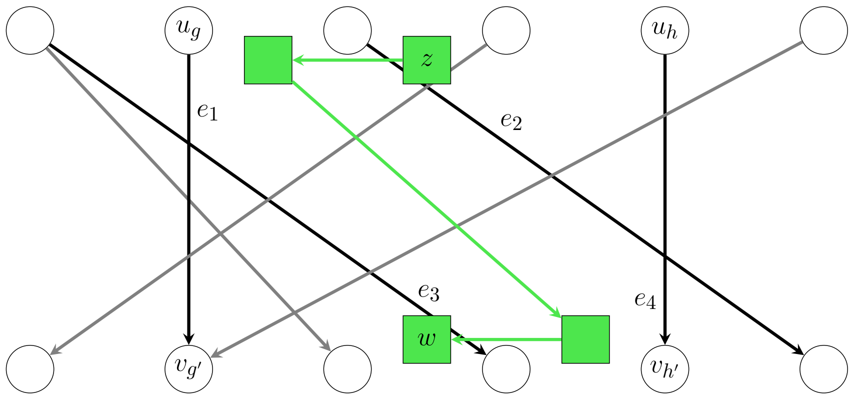


Figure 8: Case 3.1. Only relevant network nodes and edges are depicted. Edges that participate in the hug are black.

by the $\overrightarrow{bw}$. Again there are no edges that are crossed over twice by P' due to the choice of e_1 to e_4. And there are no edges in E' that are not covered by some edge of P'.

Case 3.2: *start*(e_ℓ) is the first node to the left of z with outgoing edges, but there is at least one node u' with outgoing edges and *start*$(e_r) > u' > z$.
This case is analogous to Case 3.1.

Case 3.3: *start*(e_ℓ) is the first node to the left of z with outgoing edges and *start*(e_r) is the next node to the right of z with outgoing edges.
Let $e_\ell = (u_g, v_{g'})$ and $v_{h'}$ be the first node right of $v_{g'}$ with an adjacent node $u_h > u_g$. Again we have a hug. Set $e_1 = e_\ell$, $e_2 = e_r$, e_3 to an incoming edge of $v_{h'-1}$ (or a lower node, if necessary) and $e_4 = (u_h, v_{h'})$.

With the same arguments as in Case 3.1 we convince ourselves that e_1, e_2, e_3 and e_4 are indeed a hug and we have $c_{j_i(h'-1)}$. We set $P' = z \rightarrow w_{h'-1} \overset{*}{\rightarrow} w$. As before *cost*$(P') = |E'|$.

□

Theorem 1 *Given a graph $G = (V, E)$ together with a proper layering $\mathcal{L}$ and an ordering ord, a minimum cost flow in a network with $\mathcal{O}(|V|)$ nodes and $\mathcal{O}(\min\{|V|^2, |V| + |E| + cross\})$ edges, where cross is the number of crossings in G with the layering $\mathcal{L}$ and the ordering ord, solves $HCAP_{minEL}$, i.e. computes a drawing with minimum total horizontal edge length. The cost of each network edge is in $\mathcal{O}(|E|)$.*

Proof: The previous lemmas prove the first statement. The cost of a network edge depends on the number of graph edges it crosses and is therefore in $\mathcal{O}(|E|)$.

□

Further constraints can be modelled by manipulating the network. By adjusting the lower and upper bounds of edges $a \in A$ we can realize minimum

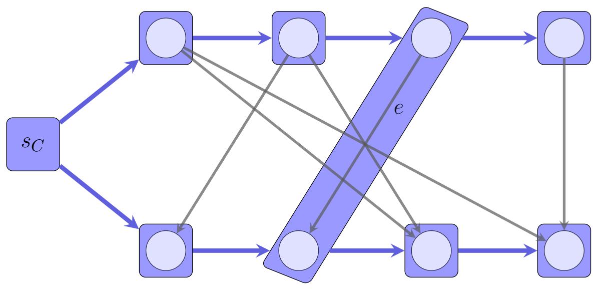


Figure 9: Constraint graph for computing the minimum width with underlying graph. The constraint graph is shown in blue. Edge e will be drawn as a vertical line.

and maximum distances between two neighboring nodes on the same layer. By removing every $g \in \overleftarrow{E}(e) \cup \overrightarrow{E}(e)$ from the network, we can enforce the edge e to be drawn vertically. In the next subsection, we explain why we are also able to minimize the total edge length while complying with a given width.

3.3 Width-Minimum Drawings with minimum Total Horizontal Edge Length

In order to produce hierarchical drawings with minimum width and shortest total horizontal edge length among all width-minimum drawings with our minimum cost flow approach, we first need to precompute the smallest possible width. This can be done by computing longest paths in an auxiliary graph C, see Figure 9. The ordering of G is modeled with C and computing longest paths corresponds to pushing every node $v \in V$ as far to the left as possible.

We construct the graph C as follows. For every vertex $v \in G$ we have a vertex $v' \in C$. If $u < v$ in G we add a directed edge (u', v') to C (transitive edges can be omitted). The cost of these edges equals the required minimum distance between the nodes of G. Finally, we add a source node s_C to C and connect it to the first node in each layer with an edge with cost zero. This source represents the left border of the drawing. If a graph edge (u, v) should be drawn as a vertical line, we merge the corresponding nodes u' and v' in C. The merged nodes guarantee that the two – or more – original nodes in G get the same x-coordinate.

The auxiliary graph C is acyclic by construction. We add edges according to the ordering of G and we only merge nodes of different layers. So if the constraints for vertical edge segments are not contradicting, i.e. two segments that cross should both be drawn vertically, we do not create any cycle in C. Hence a longest simple path to every node of C exists. By traversing the vertices of C and updating the distances to successive nodes in topological order the longest paths can be found in linear time [3].

The graph C is a subgraph of the auxiliary graph from [12] with which a coordinate assignment with minimum total horizontal edge length is computed. Gansner et al. [12] observe that assigning x-coordinates to the nodes of G can be viewed as assigning layers in the auxiliary graph, using the x-coordinates as layers. With unit minimum distances between neighboring nodes we can omit the edge costs and the source node and run the longest-path layering algorithm to get a coordinate assignment with minimum width.

Lemma 5 *The distances of longest paths from s_C to all other nodes $u \in C$ induce a solution to $HCAP_{minW}$.*

Proof: We set $x(v) = dist(v')$ for all $v \in V$, where v' is the vertex in C corresponding to v and $dist(v')$ is the distance of a longest path from s_C to v'. Let v_j and v_{j+1} be two neighboring nodes within the same layer and let $\delta(v_j, v_{j+1})$ be the required minimum distance between them. Then the longest path distances guarantee a valid x-coordinate assignment, since $x(v_{j+1}) = dist(v'_{j+1}) \geq dist(v'_j) + \delta(v_j, v_{j+1}) > dist(v'_j) = x(v_j)$. If a node v is assigned a smaller x-coordinate than $dist(v')$ then at least one of the minimum distance constraints between some nodes that lie on the longest path to v must be violated. Therefore a coordinate assignment with a smaller width than induced by the longest paths would not be valid.

□

To control the maximum width of a layered drawing we make use of Property 3, which states that the width of the drawing is at most the flow leaving s. We can add an additional node s' and an edge (s, s') to the network from the previous section and replace all edges of the form (s, w_j^1) with (s', w_j^1). Now we can limit the maximum width of the drawing by setting the upper bound of (s, s') to an appropriate value.

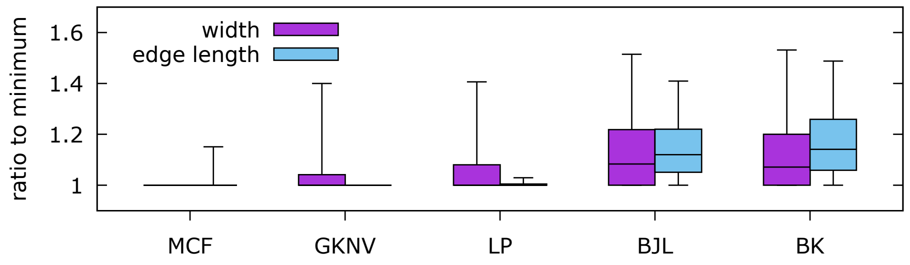
Theorem 2 *Given a graph G together with a proper layering $\mathcal{L}$ and an ordering ord, a minimum cost flow in a network with $\mathcal{O}(|V|)$ nodes and $\mathcal{O}(\min\{|V|^2, |V| + |E| + cross\})$ edges, where cross is the number of crossings in G with the layering $\mathcal{L}$ and the ordering ord, solves $HCAP_{minW\text{-}EL}$, i.e. computes a width-minimum drawing with minimum total horizontal edge length. The cost of each network edge is in $\mathcal{O}(|E|)$.*

4 Experimental Results

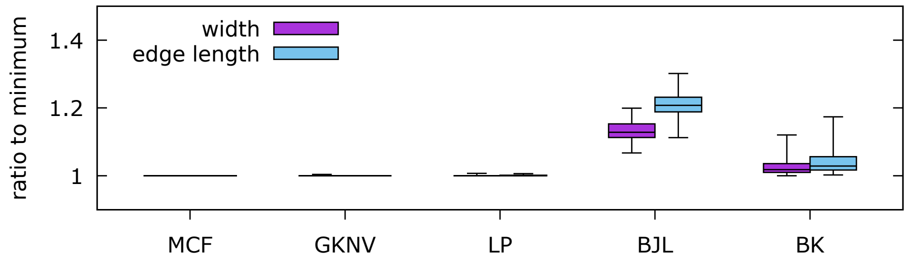
With our experiments we want to demonstrate that we are able to restrict the width of the drawing without paying too much in terms of total (horizontal) edge length and time.

We implemented the algorithm from Section 3[1], which we will call MCF, within the Open Graph Drawing Framework [7] (OGDF) and used the OGDF

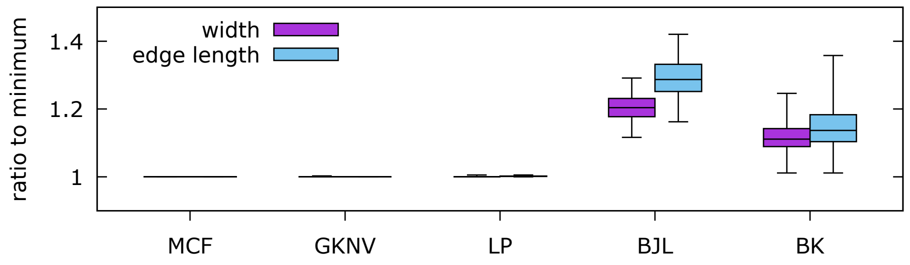
[1]available at `https://informatik.uni-koeln.de/public/spisla/CoordAssign.zip`



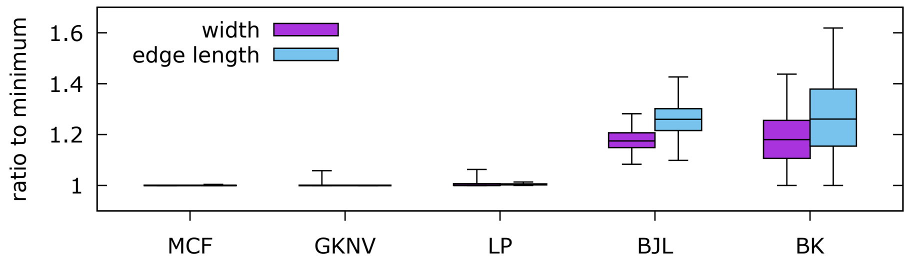
(a)



(b)

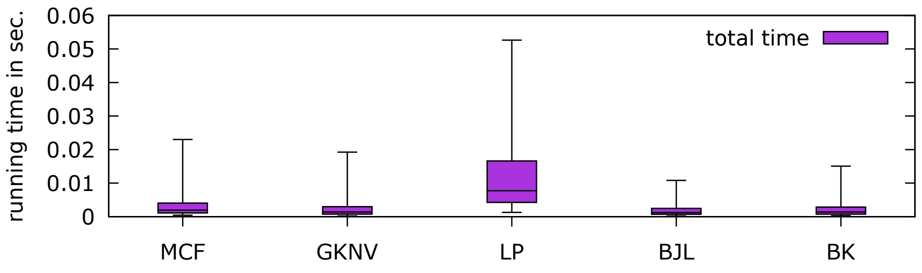


(c)

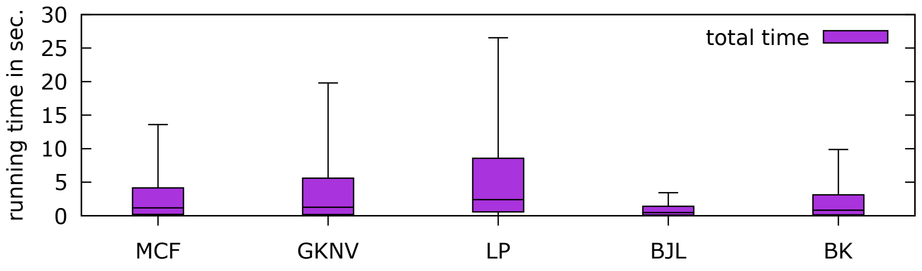


(d)

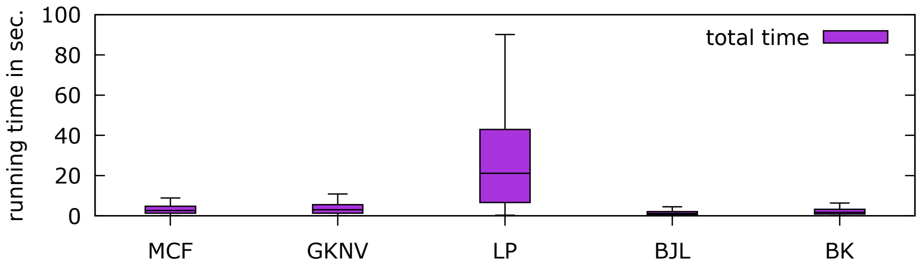
Figure 10: Relative width and total horizontal edge length.
(a) North, (b) DAGmar, (c) Large and (d) Sparse.



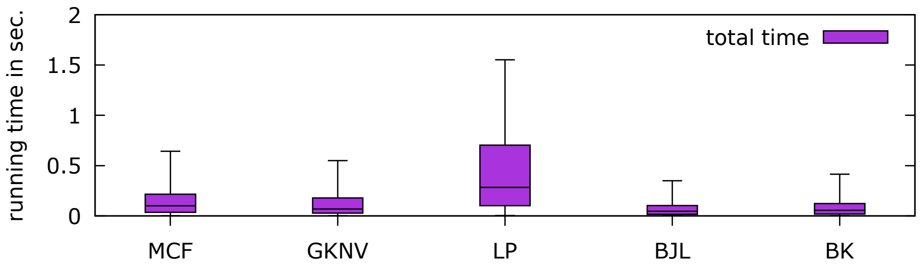
(a)



(b)



(c)



(d)

Figure 11: Running time. (a) NORTH, (b) DAGMAR, (c) LARGE and (d) SPARSE.

network simplex software to solve the minimum cost flow problem. The implementation includes the prior computation of the minimum width (see Section 3.3), which is then used to compute width-minimum drawings with minimum total edge length. We also implemented the approach of Gansner et al.[2] [12] (GKNV) that also uses the network simplex algorithm. This algorithm produces drawings with minimum total horizontal edge length, with no control of the width. Additionally we used three other OGDF methods: an ILP that also takes balancing the nodes between their neighbors into account (LP), the algorithm of Buchheim, Jünger and Leipert [6] (BJL) and the algorithm of Brandes and Köpf [4] (BK). All algorithms draw inner segments of long edges as vertical lines. We used four test sets:

- North: A subset of the AT&T graphs from `www.graphdrawing.org/data.html` consisting of 1277 graphs with 10 to 100 nodes.
- DAGmar: 1960 uniformly generated level graphs with 20 to 400 vertices and varying density (the number of edges divided by the number of nodes) from 1.6 to 10.6 [2], also available at `www.graphdrawing.org/data.html`.
- Large: To analyze the behaviour of the algorithms on larger graphs we generated 600 random level graphs with the OGDF. The graphs have between 500 and 1000 nodes and densities from 1.6 to 10.6.
- Sparse: Since level graphs from applications seem to have few edges (see e.g. experiments in [19]) we also randomly generated 2500 relatively sparse graphs with 20 to 500 nodes and densities ranging from 1.2 to 3.0.

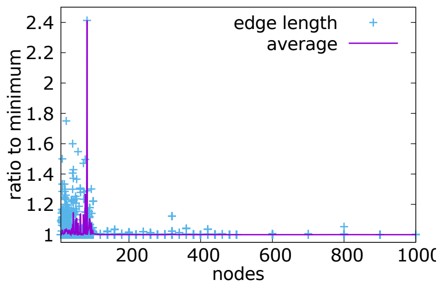
The test was run on an Intel Xeon E5-2640v3 2.6GHz CPU with 128 GB RAM. We investigated the relation of the output width to the minimum width, the relation of the output total horizontal edge length to the minimum edge length (computed by GKNV) and the running time. Figures 10 to 13, 18 and 19 show the results and Figures 14 to 17 display example drawings. The whiskers in Figures 10 and 11 cover 95% of the data and outliers are omitted for better readability.

4.1 Geometric Criteria

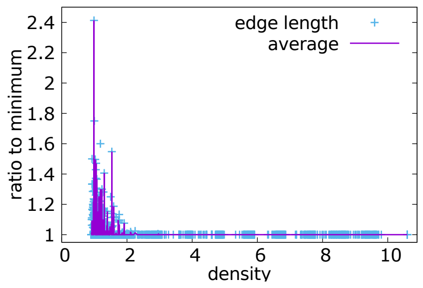
Figure 10 shows the computed width and total horizontal edge length relative to the optimum for the five algorithms and the four test sets. We can see that the total horizontal edge length of drawings produced with MCF is still close to the optimum, even though MCF has the restriction of meeting the minimum width. The average deviation of the total edge length to the optimum is 0.5% over all instances. The maximum difference to the optimum is 141.3%.

On the other hand GKNV computes drawings with near-optimum width, although the width is not explicitely targeted by the algorithm. On average

[1]also available at `https://informatik.uni-koeln.de/public/spisla/CoordAssign.zip`



(a)

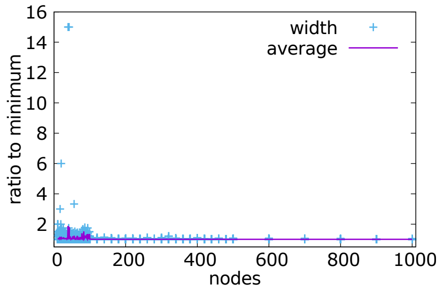


(b)

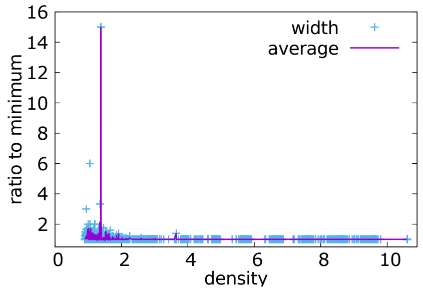
Figure 12: Ratio of the total horizontal edge length of MCF drawings to possible minimum depending on (a) the number of nodes and (b) the density.

the drawings of GKNV are 2.1% wider than width-minimum drawings. In an extreme example with minimum width 1, GKNV results in width 15.

Except for the NORTH set, there seems to be no significant difference between MCF and GKNV concerning the total horizontal edge length and the width. Figures 12 and 13 show the total edge length computed by MCF and the width computed by GKNV relative to the optimum depending on the number of nodes and the density of the input graphs. Again we see that for most of the test instances minimizing one criterion also leads to good results in the other one.



(a)



(b)

Figure 13: Ratio of the width of GKNV drawings to possible minimum depending on (a) the number of nodes and (b) the density.

Now we take a closer look at the outliers. The instances on which GKNV performs worst (9 graphs, where the computed width is more than twice as large as the optimum width) consist of a long path with some small parallel structure, see Figure 14, whereas graphs that lead to outputs of MCF with horizontally long edges (5 graphs with a more than 1.5 larger total horizontal edge length compared to the optimum) have a tree-like structure, see Figure 15. But the performance of both algorithms does not only depend on properties

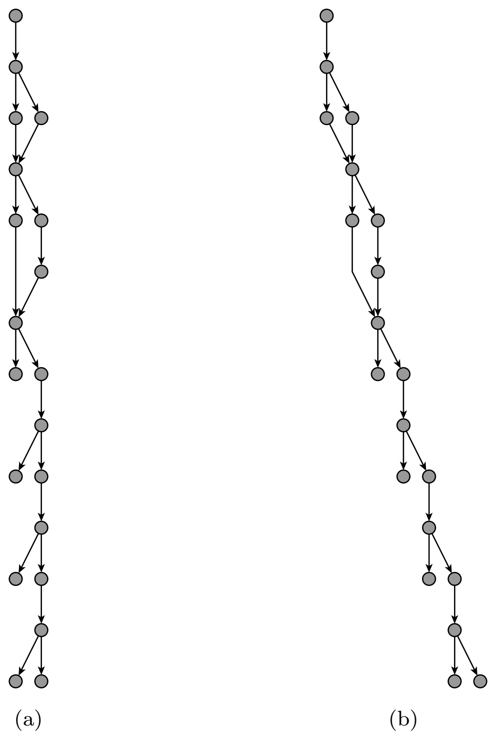


Figure 14: Example drawings of a North graph with 20 nodes and 21 edges. (a) MCF: width: 1, edge length: 8. (b) GKNV: width: 6, edge length: 8.

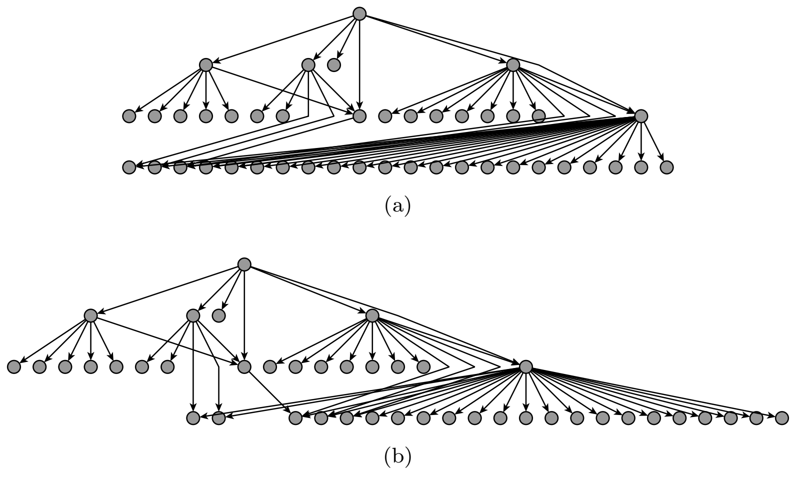


Figure 15: Example drawings of a North graph with 43 nodes and 51 edges. (a) MCF: width: 21, edge length: 352. (b) GKNV: width: 30, edge length: 220.

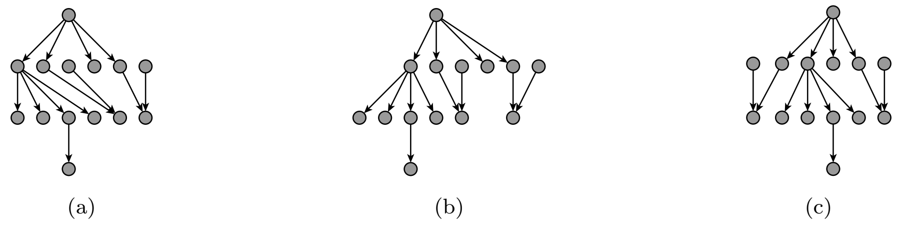


Figure 16: Example drawings of a North graph with 14 nodes and 13 edges. (a) MCF: width: 5, edge length: 18. (b) GKNV: width: 7, edge length: 12. (c) hand-made ordering and coordinate assignment: width:5, edge length 10.

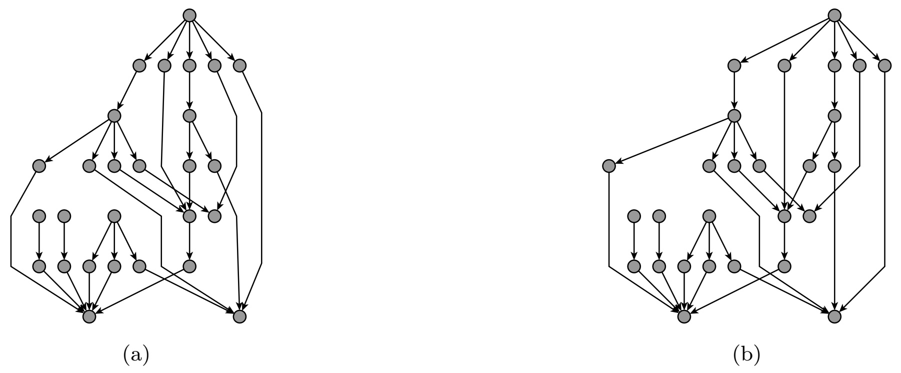


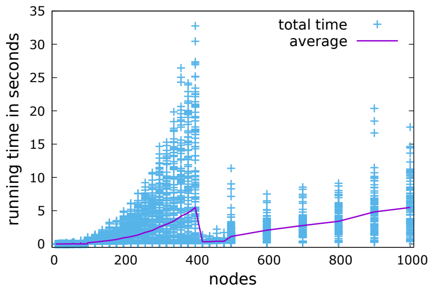
Figure 17: Example drawings of a North graph with 27 nodes and 34 edges. (a) MCF: width: 10, edge length: 49. (b) GKNV: width: 11, edge length: 48.

of the input graphs. The decisions made in the Sugiyama framework in prior phases highly influence the outcome of the coordinate assignment phase. In Figure 16 we see a graph for which neither MCF nor GKNV achieved both, the minimum width and the minimum horizontal edge length. Figure 16(c) depicts a width-minimum drawing of the same graph with a hand-made ordering that allows a smaller total edge length than GKNV computed. Figure 17 shows the drawings for graph that has neither a path-like nor a tree-like structure.

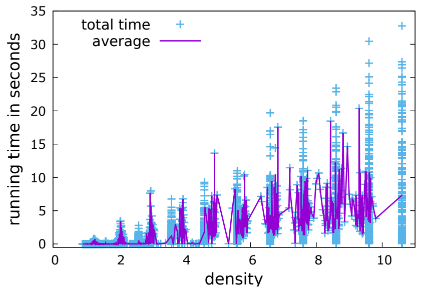
4.2 Running Time

Figure 11 displays the running time of the five algorithms and the four test sets. The heuristics BJL and BK are the fastest (0.5 seconds and 1.0 seconds on average over all instances), but MCF and GKNV are only a bit slower (1.4 seconds and 1.8 seconds). The computation time for the DAGmar and Large test sets are significantly higher than the time needed for North and Sparse.

Figure 18 shows the absolute running time of MCF over all instances depend-

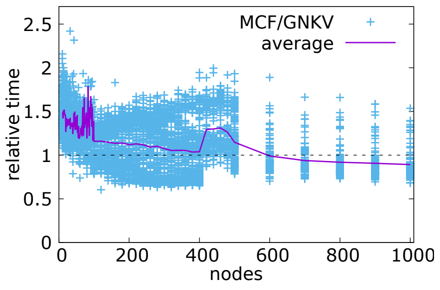


(a)

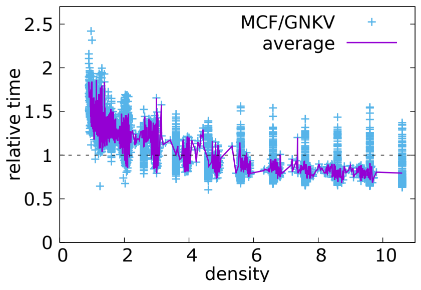


(b)

Figure 18: Absolute running time for MCF over all instances depending on (a) the number of nodes and (b) the density.



(a)



(b)

Figure 19: Ratio of the running time of MCF to the running time of GKNV depending on (a) the number of nodes and (b) the density.

ing on the number of nodes and the density of the input graphs. The longest running time is 32.8 seconds.

Figure 19 shows the relative running time of MCF compared to the time GKNV takes. In Figure 19(a) the relative running time is plotted against the number of nodes and in Figure 19(b) against the density of the input instances. On average the computation time of MCF is 17.6% longer that the computation time of GKNV. MCF is at most roughly 2.5 times slower than GKNV and takes at best 60.4% of the time needed by GKNV. We see that for larger and denser graphs MCF is faster than GKNV.

5 Conclusion

We presented a minimum cost flow formulation for the coordinate assignment problem that minimizes the total edge length with respect to several optional criteria like the maximum width or lower and upper bounds on the distance of neighboring nodes in a layer. In our experiments we showed that our approach can compete with state-of-the-art algorithms.

Future research can involve investigating how the flow approach presented here could be used or adjusted for compacting graph drawings in other drawing styles. Flow-based methods for the compaction of orthogonal drawings were adopted from VLSI design and are very efficient [17]. Since our flow technique does not affect the relative positions of the nodes to each other, it might be suitable for planar drawings.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows - theory, algorithms and applications.* Prentice Hall, 1993.

[2] C. Bachmaier, A. Gleißner, and A. Hofmeier. Dagmar: Library for dags. Technical Report MIP-1202, University of Passau, Germany, 2012. URL: `https://www.infosun.fim.uni-passau.de/~chris/down/MIP-1202.pdf`.

[3] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Monographs in Mathematics. Springer, 2008.

[4] U. Brandes and B. Köpf. Fast and simple horizontal coordinate assignment. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001*, volume 2265 of *LNCS*, pages 31–44. Springer, 2001. `doi:10.1007/3-540-45848-4_3`.

[5] J. Branke, S. Leppert, M. Middendorf, and P. Eades. Width-restricted layering of acyclic digraphs with consideration of dummy nodes. *Information Processing Letters*, 81(2):59–63, 2002. `doi:10.1016/S0020-0190(01)00200-9`.

[6] C. Buchheim, M. Jünger, and S. Leipert. A fast layout algorithm for k-level graphs. In J. Marks, editor, *Graph Drawing, 8th International Symposium, GD 2000*, volume 1984 of *LNCS*, pages 229–240. Springer, 2000. `doi:10.1007/3-540-44541-2_22`.

[7] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The open graph drawing framework (OGDF). In R. Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 543–569. Chapman and Hall/CRC, 2013.

[8] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, 1972. `doi:10.1007/BF00288685`.

[9] W. W.-M. Dai and E. S. Kuh. Global spacing of building-block layout. In *Proceedings of the IFIP International Conference on Very Large Scale Integration, VLSI'87*, pages 193–205, 1987.

[10] P. Eades, X. Lin, and R. Tamassia. An algorithm for drawing a hierarchical graph. *Int. J. Comput. Geometry Appl.*, 6(2):145–156, 1996. `doi:10.1142/S0218195996000101`.

[11] M. Forster. Applying crossing reduction strategies to layered compound graphs. In *Graph Drawing, 10th International Symposium, GD 2002*, volume 2528 of *LNCS*, pages 276–284. Springer, 2002. `doi:10.1007/3-540-36151-0_26`.

[12] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *Software Engineering*, 19(3):214–230, 1993. doi:10.1109/32.221135.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[14] P. Healy and N. S. Nikolov. A branch-and-cut approach to the directed acyclic graph layering problem. In S. G. Kobourov and M. T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002*, volume 2528 of *LNCS*, pages 98–109. Springer, 2002. doi:10.1007/3-540-36151-0_10.

[15] P. Healy and N. S. Nikolov. Hierarchical drawing algorithms. In R. Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 409–453. Chapman and Hall/CRC, 2013.

[16] A. Jabrayilov, S. Mallach, P. Mutzel, U. Rüegg, and R. von Hanxleden. Compact layered drawings of general directed graphs. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD 2016*, volume 9801 of *LNCS*, pages 209–221. Springer, 2016. doi:10.1007/978-3-319-50106-2_17.

[17] G. W. Klau, K. Klein, and P. Mutzel. An Experimental Comparison of Orthogonal Compaction Algorithms (Extended Abstract). In J. Marks, editor, *Graph Drawing, 8th International Symposium, GD 2000*, volume 1984 of *LNCS*, pages 37–51. Springer, 2000. doi:10.1007/3-540-44541-2_5.

[18] L. Nachmanson, G. G. Robertson, and B. Lee. Drawing graphs with GLEE. In S. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing, 15th International Symposium, GD 2007*, volume 4875 of *LNCS*, pages 389–394. Springer, 2007. doi:10.1007/978-3-540-77537-9_38.

[19] U. Rüegg, C. D. Schulze, J. J. Carstens, and R. von Hanxleden. Size- and port-aware horizontal node coordinate assignment. In E. Di Giacomo and A. Lubiw, editors, *Graph Drawing and Network Visualization - 23rd International Symposium, GD 2015*, volume 9411 of *LNCS*, pages 139–150, 2015. doi:10.1007/978-3-319-27261-0_12.

[20] G. Sander. A fast heuristic for hierarchical manhattan layout. In F. Brandenburg, editor, *Graph Drawing, Symposium on Graph Drawing, GD '95*, volume 1027 of *LNCS*, pages 447–458. Springer, 1995. doi:10.1007/BFb0021828.

[21] 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, Feb. 1981. doi:10.1109/TSMC.1981.4308636.