Straightening Drawings of Clustered Hierarchical Graphs*

Sergey Bereg[1], Markus Völker[2,**], Alexander Wolff[2,*], and Yuanyi Zhang[1]

[1] Dept. of Computer Science, University of Texas at Dallas, U.S.A.
{besp,yzhang}@utdallas.edu
[2] Fakultät für Informatik, Universität Karlsruhe, Germany
mvoelker@ira.uka.de
http://i11www.ira.uka.de/people/awolff

Abstract. In this paper we deal with making drawings of clustered hierarchical graphs nicer. Given a planar graph $G = (V, E)$ with an assignment of the vertices to horizontal layers, a plane drawing of G (with y-monotone edges) can be specified by stating for each layer the order of the vertices lying on and the edges intersecting that layer. Given these orders and a recursive partition of the vertices into clusters, we want to draw G such that (i) edges are straight-line segments, (ii) clusters lie in disjoint convex regions, (iii) no edge intersects a cluster boundary twice.

First we investigate fast algorithms that produce drawings of the above type if the clustering fulfills certain conditions. We give two fast algorithms with different preconditions. Second we give a linear programming (LP) formulation that always yields a drawing that fulfills the above three requirements—if such a drawing exists. The size of our LP formulation is linear in the size of the graph.

1 Introduction

A graph is often associated with structural information that needs to be made explicit when drawing the graph. There are many ways in which structure can be given, but usually it comes in one of two ways: clusters or hierarchies. A clustering of a graph is a (possibly recursive) partition of the vertex set into so-called clusters. The vertices in the same cluster are interpreted as being similar or close, those in different clusters as different or far from each other in some sense. It is common to visualize disjoint clusters by placing their vertices in disjoint convex regions. For example in the Ptolemy II project (heterogeneous modeling, simulation, and design of concurrent systems), clustered graphs are used to represent (possibly nested) parts of embedded systems, see Fig. 1. Hierarchies also partition the vertex set, but not according to proximity, but according to rank. The rank of a vertex reflects its importance or status in relationship to vertices of lower or higher rank. Usually vertices of equal rank are placed on horizontal

* A preliminary version of this work was presented as a poster at SOFSEM'06.
** Supported by grant WO 758/4-2 of the German Research Foundation (DFG).

Jan van Leeuwen et al. (Eds.): SOFSEM 2007, LNCS 4362, pp. 176–187, 2007.

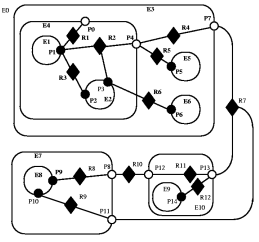


Fig. 1. A Ptolemy-II model: vertices represent actors, edges communication

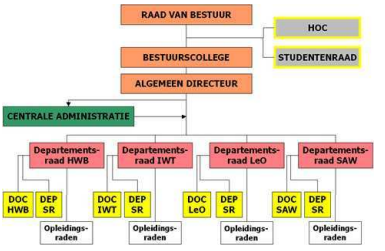


Fig. 2. The organigram of Hogeschool Limburg: vertices represent administrative entities, edges interaction

lines, to which we refer as *layers*. Examples of hierarchical graphs are so-called organigrams that are used to represent the structure of organizations, see Fig. 2. For both clustered and hierarchical graphs there is an abundance of literature. Brockenauer and Cornelsen give an overview [1].

In this paper we deal with graphs that have both clusters and a hierarchy. This makes it possible to visualize two different graph structures at the same time. The challenging question is how to overlay these different and independent structures. We model the problem as follows. Given a planar graph $G = (V, E)$ with an assignment of the vertices to horizontal layers, a plane drawing of G (with polygonal or y-monotone edges) can be specified by stating for each layer the order of the vertices lying on and the edges intersecting that layer. Given these orders and a recursive partition of the vertices into clusters, our aim is to draw G such that (i) edges are straight-line segments, (ii) clusters lie in disjoint convex regions, and (iii) no edge intersects a cluster boundary twice.

Our first contribution consists of two fast algorithms that draw clustered hierarchical graphs if certain preconditions are met. Both algorithms require that the left-to-right ordering of the clusters is *consistent*, i.e., the precedence relationship of the clusters is the same over all layers. The first algorithm runs in $O(n^2)$ time and additionally relies on the *cluster adjacency graph* (to be defined later) being acyclic, see Section 3. The second algorithm runs in linear time and requires that clusters can be separated by y-monotone paths, see Section 4. The preconditions for both algorithms can be tested in linear time.

Our second contribution is a linear programming (LP) formulation that always yields a drawing if a drawing with straight-line edges and non-intersecting convex cluster regions exists, see Section 5. The number of variables and constraints in our LP formulation is linear in the size of the graph. If either of the above-mentioned constraints is satisfied, the existence of the corresponding algorithm shows that the LP formulation also yields a drawing. The LP is obviously less efficient than the above algorithms, but it is more general, more flexible, and yields nicer results due to global optimization. The LP allows the user to incorporate esthetic criteria. For example, one can use additional constraints to enforce minimum vertex-vertex distances. We also suggest two different objective

functions; one minimizes the width of the drawing, the other tries to maximize the angular resolution. The LP can also draw non-planar graphs; it keeps exactly the crossings of the input graph. We extend the basic LP to be able to process rectangular vertices as in Fig. 2. We have implemented the LP and applied it to a number of planar and non-planar graphs, see Fig. 10. Our implementation can be tested via a Java applet under the URL `http://i11www.ira.uka.de/clusteredgraph/`.

Our work builds on the seminal work of Eades et al. [2]. They define a clustered graph to be *compound planar* (c-planar) if it admits a drawing with no edge crossings or edge-region crossings, where the regions are the convex hulls of the clusters. They present an algorithm that draws clustered *c-plane* graphs, i.e., c-planar graphs given a c-planar embedding. (An embedding is defined by the counter-clockwise order of the edges incident to a vertex and by specifying the outer face.) From the embedding they compute a special st-numbering, the so-called c-st numbering, which maps each vertex v of G to a unique layer $\lambda(v)$, i.e., an integer y-coordinate from the set $\{1, \ldots, n\}$. The layer assignment is such that the vertices that belong to the same cluster occupy consecutive layers. The assignment is then used to draw the graph as a hierarchical graph with straight-line edges. Since each cluster occupies a range of consecutive layers, the convex hulls of different clusters do not intersect. Moreover, since each cluster is assumed to be connected and the algorithm for drawing hierarchical graphs does not produce edge crossings, no edge intersects a cluster hull more than once.

A draw-back of the algorithm of Eades et al. for drawing clustered graphs is that it produces a drawing of height n for any n-vertex graph. For example, it draws the graph of a $k \times k$ square grid on k^2 horizontal lines, although this graph can easily be drawn on k lines. Eades et al. list vertical compaction among the important problems for further research concerning the drawing of clustered graphs. Vertical compaction can be divided into two steps: (a) assign vertices to layers and (b) draw the hierarchical graph. This paper deals with step (b).

Concerning the first step Bachmeier and Forster [3] have shown how to check in $O(kn)$ time whether a graph has a planar k-level embedding. If an embedding exists, it is computed within the same time bound. However, they restrict the input to *proper layer-connected single-source* graphs. A hierarchical graph is *proper* if no edge crosses a layer, i.e., if $|\lambda(u) - \lambda(v)| = 1$ for every edge uv. A clustered hierarchical graph is *layer-connected* if in each cluster each pair of consecutive layers is spanned by an edge of the cluster. A *source* is a vertex that is only connected to vertices of higher levels.

Rectangular cluster regions and *rectilinear* edges were considered by Sugiyama and Misue [5] and by Sander [4]. They give algorithms for drawing *compound* graphs, which generalize clustered graphs in that edges between clusters or between clusters and vertices are also allowed. Both algorithms extend the classical algorithm of Sugiyama et al. [6] for drawing hierarchical graphs. Like Eades et al. [2], Sugiyama and Misue [5] place each vertex on a separate horizontal level, while Sander [4] tries to produce more compact drawings.

2 Preliminaries

A *clustered graph* $\mathcal{C} = (G, T)$ consists of an undirected graph $G = (V, E)$ and a rooted tree $T = (V_T, E_T)$ such that the leaves of T are in one-to-one correspondence with the vertices of G. A subset C of V is called a *cluster* if C is the set of leaves of the subtree rooted at a vertex of V_T. A *drawing* of a graph $G = (V, E)$ assign positions $\pi : V \to \mathbb{R}^2$ to the vertices of V and to each edge $(u, v) \in E$ a simple Jordan curve joining $\pi(u)$ and $\pi(v)$. A drawing is *planar* if the curves of different edges do not cross. We say that a drawing is *weakly monotone* if all curves are weakly monotone in y-direction, i.e., for each curve it holds that its intersection with a horizontal line is empty or connected. For *strictly monotone* the intersection must be empty or a point. In other words: we allow horizontal edges between neighboring vertices on the same layer. A special case of monotone drawings are *straight-line drawings*, where all curves are straight-line segments.

A *layered* or *hierarchical* graph $\mathcal{L} = (G, \lambda)$ is given by a graph $G = (V, E)$ and an assignment $\lambda : V \to \{1, \ldots, k\}$ of the vertices to horizontal layers $y = 1, \ldots, y = k$. For a hierarchical graph we define V_i to be the set of vertices on level i, i.e., $V_i = \{v \in V \mid \lambda(v) = i\}$ and E_i to be the set of edges crossing level i, i.e., $E_i = \{\{u, v\} \in E \mid (\lambda(u) - i)(\lambda(v) - i) < 0\}$. A monotone drawing D of G induces the *x-order* of $V_i \cup E_i$, i.e., a bijection $\lambda_i : V_i \cup E_i \to \{1, 2, \ldots, n_i\}$ where $n_i = |V_i \cup E_i|$. The layer assignment λ and the x-orders $\lambda_1, \ldots, \lambda_k$ induced by D yield another monotone drawing D' of G, where each edge $e = (u, v)$ is represented by a polygonal chain, namely the chain given by the point sequence $(\lambda_i(e), i), (\lambda_{i+1}(e), i + 1), \ldots, (\lambda_j(e), j)$, where $i = \min\{\lambda(u), \lambda(v)\}$ and $j = \max\{\lambda(u), \lambda(v)\}$. Note that D' is plane if and only if D is plane.

In this paper we assume that we are given a clustered hierarchical c-plane graph (G, T, λ) including the x-orders $\lambda_1, \ldots, \lambda_k$ of a monotone planar drawing of G. Our aim is to investigate conditions under which we can efficiently determine a straight-line drawing of G that respects the x-orders and has convex cluster regions.

Eades at al. [2] have given a linear-time algorithm that draws clustered c-plane graphs such that edges are drawn straight and cluster regions are convex. The main disadvantage of that algorithm is that it places each vertex on a unique layer. Eades et al. require that the curves in the given drawing are strictly y-monotone and that the subgraph induced by each cluster is connected. We only require weak monotonicity. The layer assignment that Eades et al. compute has the property that the vertices of each cluster are assigned to a set of consecutive layers. We do not require this. However, we require that the x-orders λ_i are *consistent*, i.e., for any pair of clusters C and C' and any pair of layers i and j it holds that if $\lambda_i(v) < \lambda_i(v')$ then $\lambda_j(w) < \lambda_j(w')$ for all $v, w \in C$ and $v', w' \in C'$.

3 Recursive Algorithm

In this section we make a stronger assumption on the x-orders of the vertices on each layer. Let F be the directed graph whose vertices correspond to clusters

and, for two clusters C and C', there is an edge (C, C') if there is a level i with $\lambda_i(t) < \lambda_i(t')$, where t is either a vertex of C or an edge incident to C and t' is either a vertex of C' or an edge incident to C'. If F, the *cluster adjacency graph*, is acyclic, we say that the layer assignment λ is *strongly consistent*. Note that F is planar since G is c-planar.

A c-plane clustered graph with strongly consistent layer assignment can be triangulated in linear time such that the same layer assignment is strongly consistent in the resulting graph. We show that every triangulated hierarchical plane graph with strongly consistent layer assignment admits a straight-line drawing with a prescribed external face that is the complement of a convex polygon P, i.e., $\mathbb{R}^2 \setminus P$. We borrow some terminology from Eades et al. [2]. As Eades at al. we allow slightly more general polygons than convex polygons. Strictly speaking, in a convex polygon each vertex has an interior angle of less than 180°. We call such a vertex an *apex*. We also allow *flat* vertices where the two incident edges form an angle of 180°. When we map vertices of the given c-plane graph G to those of the polygon P we must be careful with these flat vertices. We say that a polygon P is *feasible* for G if (i) P is a convex polygon, and (ii) if abc is a face of G and the vertices a, b, and c are vertices of P, then they are not collinear.

It is easy to construct a feasible polygon for the graph G; for example, all the vertices of the outer face can be made apexes of the polygon. We present a recursive procedure to draw the graph G. Consider the cluster adjacency graph F defined above. Since F is acyclic, F has a *topological* ordering, i.e., an ordering $C_1, C_2, \ldots$ of the clusters such that $i < j$ for any edge (C_i, C_j) of F. Note that F has linear size and can thus be sorted topologically in time linear in the number of vertices, i.e., clusters. The first cluster C_1 has in-degree 0 in F. We split G into the subgraph G_1 induced by the vertex set V_1 of C_1 and a subgraph G_2 induced by $V_2 = V \setminus V_1$. Color the vertices in V_1 black and those in V_2 white. Let edges with a white and a black endpoint be gray. Due to our choice of C_1 there are exactly two gray edges e and e' on the outer face. These are connected by a path of inner faces each of which has exactly two gray edges.

Now we split the polygon P into two polygons by a line ab in order to treat G_1 and G_2 recursively. We choose the points a and b anywhere on the edges e and e', respectively, see Fig. 3.

Our recursion is as follows:

1. Construct feasible polygons P_1 and P_2 for G_1 and G_2 that are separated by the line ab and are consistent with P.
2. For $i = 1, 2$ draw G_i in P_i recursively.
3. Draw the gray edges as straight-line segments.

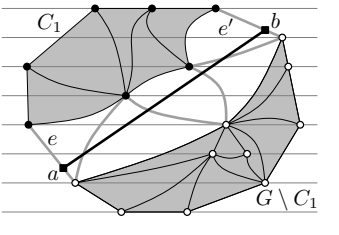
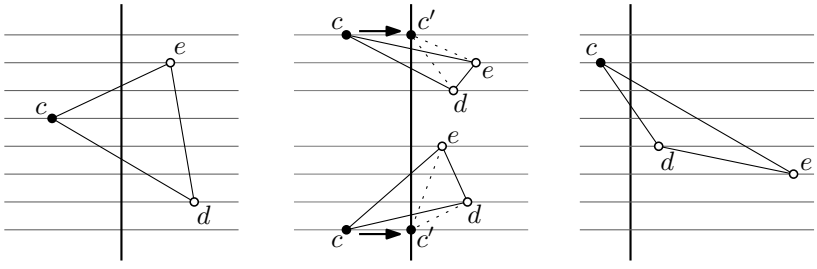


Fig. 3. Split along ab

Unfortunately, this may produce a drawing with crossing edges, see Fig. 5. We now show how this problem can be fixed by introducing dummy vertices.



(a) type A (b) type B (c) type C

Fig. 4. The three types of faces

We analyze the faces split by the line ab. Each of these faces contains at least one vertex in C_1 and at least one vertex not in C_1. Let cde be a face that is crossed by ab such that c is the only vertex in C_1. The case where C_1 contains two vertices of cde is symmetric. Without loss of generality we assume that cde is the clockwise order of the vertices. In general, there are three types of faces depending on the layer assignment of the vertices c, d, and e, see Fig. 4:

(A) $\lambda(d) \leq \lambda(c) \leq \lambda(e)$,
(B) $\lambda(d) \leq \lambda(e)$ and ($\lambda(c) \leq \lambda(d)$ or $\lambda(c) \geq \lambda(e)$), and
(C) $\lambda(e) \leq \lambda(d) \leq \lambda(c)$.

Faces of type A can be handled by the above approach. We show that there are no faces of type C. Indeed, if C_1 contains only one vertex c of a face as in Fig. 4 (c), then the edge ce crosses the layer of vertex d and the order $\lambda_i(d) < \lambda_i(ce)$ is not consistent with the x-order of layer $i = \lambda(d)$. Note that faces of type B cause a problem, see the face uvw in Fig. 5. For each type-B face abc we do two things. First, we introduce a dummy vertex c' at the intersection of the separating line ab and the layer of c, see Fig. 4 (b). Second, we add the triangle $c'de$ to the graph G_2. Then we connect each pair of consecutive dummy vertices by an edge and triangulate new faces if they contain more than three vertices. The triangulation can be done arbitrarily. We construct a feasible polygon for G_2 that contains the dummy points on ab and is consistent with P. Similarly we add vertices and faces to the graph G_1 if there are faces of type B with two vertices in C_1.

Let G_1' and G_2' be the graphs constructed as above and let P_1' and P_2' be the corresponding convex polygons. Then it is not hard to see the following.

Lemma 1 (Recursion). *The polygons P_1' and P_2' are feasible for the graphs G_1' and G_2', respectively. The positions of the vertices of V in the polygons P_1' and P_2' can be used for the straight-line drawing of G.*

At the bottom level of the recursion the graph G contains only one cluster and the linear-time drawing algorithm for hierarchical graphs by Eades at al. [2] can be used. In the full version of this paper we show the following theorem.

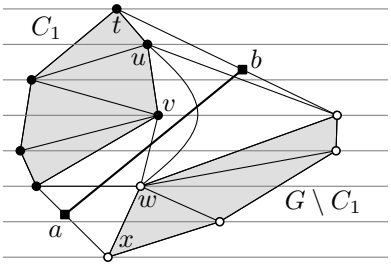


Fig. 5. After splitting along ab simple recursion does not work: edge uv and vertex w are not independent

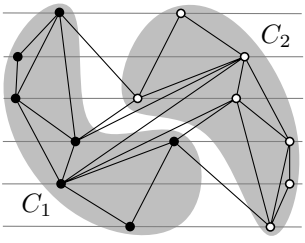


Fig. 6. A clustered hierarchical graph *without* a monotone separating path

Theorem 1 (Algorithm). *Let (G, T, λ) be a clustered hierarchical c-plane graph whose n vertices are assigned to k layers. If the layer assignment is strongly consistent, then a straight-line drawing with convex cluster regions can be computed in $O(n^2)$ time.*

4 Separating-Path Algorithm

The algorithm of Section 3 is recursive and guarantees a c-plane drawing if the layer assignment is strongly consistent. However, the layer assignment of a clustered graph may not be strongly consistent even for two clusters. Therefore we now discuss an algorithm with a different requirement. We explore the possibility of splitting the graph along a path. A *monotone separating path* in a clustered hierarchical c-plane graph $G = (V, E)$ is defined as a path Π between two vertices on the boundary of G such that (i) the path is y-monotone, and (ii) the graph $G - \Pi$ has two connected components G_1 and G_2 whose vertices are in different clusters, i.e., for any cluster C_i, $C_i \cap G_1 = \emptyset$ or $C_i \cap G_2 = \emptyset$. For example, the graph shown in Fig. 5 admits the monotone separating path $tuvwx$. Although there are clustered c-plane graphs without separating paths, see Fig. 6 (a), the requirement is intuitive and does not seem too restrictive for practical applications.

Finding a monotone separating path. Suppose that G has only two clusters. We show how to detect whether they can be separated by a monotone separating path. An edge $(u, v), \lambda(u) \leq \lambda(v)$ is *separating* if it separates the clusters in the slab $\lambda(u) \leq y \leq \lambda(v)$. The boundary of G contains exactly two edges g_1 and g_2 called *gates* whose endpoints are in different clusters. We want to find a y-monotone path Π between two vertices u_1 and u_2 such that $u_i, i = 1, 2$ is an endpoint of g_i and every edge of Π is separating.

We sweep the plane with a horizontal line l from top to bottom. We maintain a list L of edges intersecting l. An edge $e \in L$ with is *good* if its part above l satisfies the definition of the separating edge; otherwise e is called *bad*. The good and bad edges satisfy the property that the list L consists of three sublists L_1, L_2, and L_3 such that all good edges are in L_2, see Fig. 7. We just store

the first and last good edge of L_2. Suppose that l approaches layer i. In the list of vertices of layer i, we find two consecutive vertices a and b from different clusters, see Fig. 7. We proceed as follows.

1. Delete edges that end at layer i. If a good edge ending at a or b is deleted then it is a separating edge.
2. Reduce the list L_2 using the positions of a and b in L.
3. Insert new edges into L. A new edge falls into L_2 if it starts at a or b.

The sweep can be done in linear time since we maintain only the first and last edges of L_2 and the rest is just traversing a planar graph. We create a directed graph G' using separating edges by orienting them from top to bottom. Any monotone separating path in G connects two vertices that belong to different gates g_1 and g_2. A path connecting gates in G' can be found in linear time. Note that a separating path may not exist, see Fig. 6.

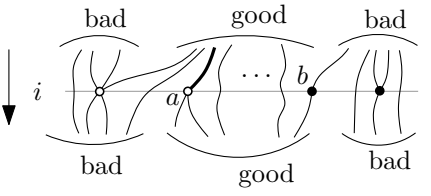


Fig. 7. Traversing layer i. Vertices a and b are consecutive vertices from different clusters. The separating edge is bold.

Shortcuts. We compute shortcuts in the separating path. There are two types of shortcuts—left and right. The shortest path using left (right) shortcuts is called *left path* (resp. *right path*). We find two graphs G_1 and G_2 using these paths, see Fig. 8 (a). We draw the left and right paths using parallel line segments and compute drawings of G_1 and G_2 using the algorithm of Eades et al. [2].

Final drawing. Let $\xi, \delta > 0$ be two parameters. We place the drawings of G_1 and G_2 at distance ξ from each other. The remaining vertices are placed on two arcs a_1, a_2 using distance δ as shown in Fig. 8 (c). The values of ξ and δ are subject to the following restrictions. Consider a face abc. If its two vertices b and c are in G_2 (or G_1), then the restriction is $\xi < \xi_0$, see Fig. 9 (a). If exactly one vertex is in G_2 (or G_1), then the restriction is $\xi < \xi_1$, see Fig. 9 (b). If a, b and c lie on the arcs a_1, a_2, then the drawing of the face abc is correct if δ is chosen small, see Fig. 9 (c). This procedure yields the following theorem.

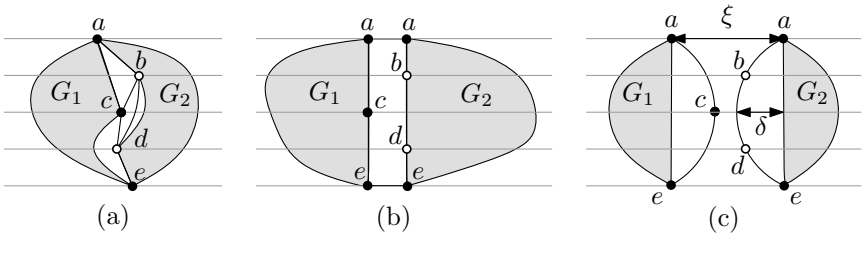


Fig. 8. (a) Shortcuts in the separating path—the left path is ace, the right path $abde$, (b) recursive drawing of G_1 and G_2, (c) the two parameters ξ and δ

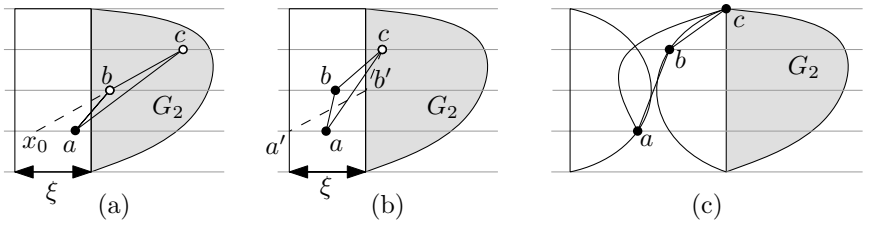


Fig. 9. Restrictions for the correct drawing of the face abc that belongs to G_1 and G_2: (a) $\xi < \xi_0$, where ξ_0 is the distance of x_0 from G_2, (b) $\xi < \xi_1$, where ξ_1 is derived from the condition that the slope of $b'c$ is less than the slope of $a'b'$, (c) $\delta > 0$ such that a is above bc

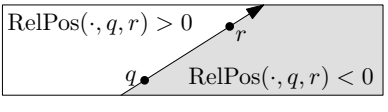
Theorem 2. *Given a clustered hierarchical c-plane graph G with two clusters and a monotone separating path, a straight-line drawing of G with convex cluster regions can be computed in linear time.*

5 Linear Programming Formulation

In this section we describe how a clustered hierarchical graph (G, T, λ) can be drawn "nicely", i.e., with straight-line edges and disjoint convex cluster regions. We give an LP formulation that decides whether the input graph has a nice drawing. Note that this is always the case if the layer assignment is strongly consistent. If a nice drawing exists, then the objective function of our LP formulation yields an especially nice drawing. A great advantage of our LP in comparison with other algorithms is that it can handle unconnected and non-planar graphs. The edge crossings of the input are preserved and no new ones are produced. In the description of our LP formulation we only consider clusters on the top level of the cluster hierarchy, i.e., children of the root of T. Clusters at lower levels can be treated analogously. We have implemented the LP. For results, see Fig. 10. Our LP can easily be adapted to more complex drawings, e.g., for graphs with labeled vertices and edges or for graphs with icons as vertices (see full version).

For three points $p = (p_x, p_y)$, $q = (q_x, q_y)$, and $r = (r_x, r_y) \neq q$ in the plane, let their *relative position* $\mathrm{RelPos}(p, q, r)$ be defined by the following determinant:

$$\mathrm{RelPos}(p, q, r) = \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}.$$



Observe that $\mathrm{RelPos}(p, q, r) > 0$ iff p lies to the left of the line from q to r. Note that these are *linear* constraints if the y-coordinates of the points are known.

5.1 Constraints

Our LP formulation has to fulfill three requirements. First, the x-orders $\lambda_1, \ldots, \lambda_k$ must be preserved. Second, the edges must be straight-line segments. Third, the convex hulls of the clusters must be disjoint.

For the first requirement we do the following. For each vertex $v \in V$ we introduce a variable x_v that will express the x-coordinate of v. Similarly, for each edge $e \in E$ and each level $y \in \{1, \ldots, k\}$ spanned by e we introduce the variable $x_{e,y}$ if the immediate predecessor or successor of e on level y is a vertex (and not another edge). Since on each level y the x-order of the vertices and the edges spanning the level is part of the input, we preserve this order by constraints

$$x_a < x_b, \tag{1}$$

where a and b are either vertices or edge-level pairs and a is the immediate predecessor of b in the x-order λ_y. We can also use these constraints to ensure a certain minimum horizontal distance $d_{\min}$ between a and b:

$$x_a + d_{\min} \leq x_b. \tag{2}$$

Since each vertex is the immediate neighbor of at most two edges, the first requirement needs $O(n)$ variables and constraints.

For the second requirement we proceed as follows. For each pair of an edge $e = \{u, w\} \in E$ and a level $y \in \{1, \ldots, k\}$ for which we have introduced the variable $x_{e,y}$ above, we now introduce the constraint

$$\mathrm{RelPos}\big((x_{e,y}, y), u, w\big) = 0. \tag{3}$$

This makes sure that the intersection point of edge e and level y lies on the straight line through u and w. Since there are $O(n)$ variables of type $x_{e,y}$, the second requirement needs $O(n)$ new constraints.

For the third requirement it is simple to come up with a solution that needs $\Theta(n^3)$ constraints. We only need $O(n)$ constraints, basically by observing that the cluster adjacency graph is planar. We introduce two new variables x_{ij} and X_{ij} for each pair (C_i, C_j) of adjacent clusters, i.e., clusters with vertices $v \in C_i$ and $w \in C_j$ where v is the immediate predecessor of w in the x-order on level $\lambda(v) = \lambda(w)$. Let $\{y_{ij}, \ldots, Y_{ij}\} = \lambda(C_i) \cap \lambda(C_j)$. The idea is to define two points $p_{ij} = (x_{ij}, y_{ij})$ and $P_{ij} = (X_{ij}, Y_{ij})$ such that the line segment from p_{ij} to P_{ij} will separate the two clusters C_i and C_j. To ensure this separation we introduce the following constraint for each vertex u with $y_{ij} \leq \lambda(u) \leq Y_{ij}$ that is rightmost in C_i, i.e., $x_u > x_{u'}$ for all $u' \in C_i$ with $\lambda(u) = \lambda(u')$:

$$\mathrm{RelPos}(p_{ij}, P_{ij}, u) < 0 \tag{4}$$

The constraint for the leftmost vertices is symmetric. Since each vertex $v \in V$ is leftmost or rightmost relative to at most two clusters, the number of constraints of this type is also linear. By construction the system of Equations (1), (3), and (4) has a solution if and only if the clustered graph can be drawn nicely.

5.2 Objective Functions

If a nice drawing exists, then we would like to choose a particularly nice one. Therefore we try to produce balanced drawings, in which the angular space of

180° above and below each vertex is distributed uniformly among the respective vertices. We treat the vertices one by one. Let v be the current vertex. For each vertex u adjacent to v an optimal position relative to v can easily be computed. For this purpose the adjacent vertices above and below v are uniformly distributed. As the vertical distances are fixed, we are able to calculate δ^*_{uv}, the optimal x-offset of u relative to v, using trigonometric functions. The actual horizontal offset δ_{uv} between u and v is given by $\delta_{uv} = x_u - x_v$. The absolute difference μ_{uv} of δ_{uv} and δ^*_{uv} can now be expressed as follows:

$$\mu_{uv} \geq +\delta^*_{uv} - \delta_{uv} \qquad \text{and} \qquad \mu_{uv} \geq -\delta^*_{uv} + \delta_{uv} \tag{5}$$

The variable μ_{uv} indicates how much the actual position of u relative to v differs from the ideal one. We normalize μ_{uv}:

$$\bar{\mu}_{uv} = \frac{\mu_{uv}}{|y_v - y_u|} \tag{6}$$

Summing up $\bar{\mu}_{uv}$ over all edges $\{u, v\} \in E$ yields the following objective function:

$$\text{minimize} \sum_{\{u,v\} \in E} (\bar{\mu}_{uv} + \bar{\mu}_{vu}) \tag{7}$$

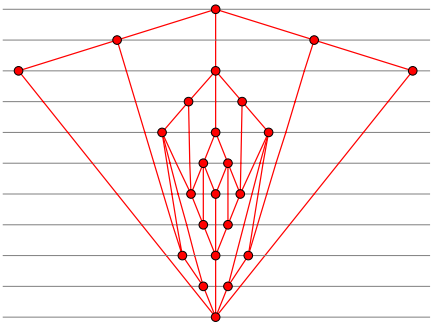
Note that in general $\bar{\mu}_{uv}$ and $\bar{\mu}_{vu}$ differ. Instead of optimizing angles, it is also possible to optimize the width of the drawing. This is achieved by

$$\mu_{uv} \geq -\delta_{uv} \qquad \text{and} \qquad \mu_{uv} \geq +\delta_{uv}. \tag{8}$$

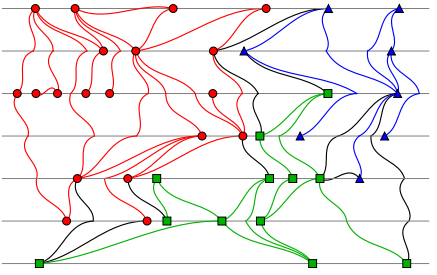
Recall that constraint (2) makes sure that the minimum distance between vertices is kept. Equation (6) and objective function (7) remain as before. For example drawings see Fig. 10. Note that graph G_2 is not plane. Also note that H_3 is not clustered; the drawing shows that our LP nicely keeps the symmetry.

References

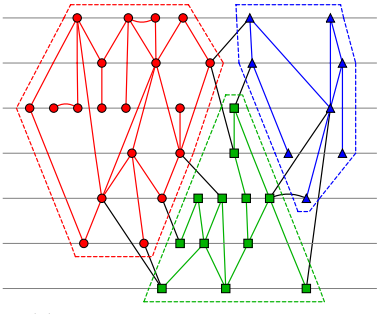
1. Brockenauer, R. and Cornelsen, S.: Drawing Clusters and Hierarchies. In M. Kaufmann and D. Wagner (eds), Drawing Graphs: Methods and Models, Springer-Verlag, Lecture Notes in Computer Science **2025** (2001) 193–227
2. Eades, P., Feng, Q., Lin, X., and Nagamochi, H.: Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs. Algorithmica **44** 1 (2005) 1–32
3. Forster, M. and Bachmaier, C.: Clustered Level Planarity. In P. van Emde Boas, J. Pokorny, M. Bielikova, and J. Stuller, (eds), Proc. 30th Int. Conf. Current Trends in Theory and Practice of Computer Science (SOFSEM'04), Springer-Verlag, Lecture Notes in Computer Science **2932** (2004) 218–228
4. Sander, G.: Graph Layout for Applications in Compiler Construction. Theoretical Computer Science **217** (1999) 175–214
5. Sugiyama, K. and Misue, K.: Visualization of Structural Information: Automatic Drawing of Compound Digraphs. IEEE Transactions on Systems, Man, and Cybernetics **21** 4 (1991) 876–891
6. Sugiyama, K., Tagawa, S., and Toda, M.: Methods for Visual Understanding of Hierarchical System Structures. IEEE Transactions on Systems, Man, and Cybernetics **11** 2 (1981) 109–125



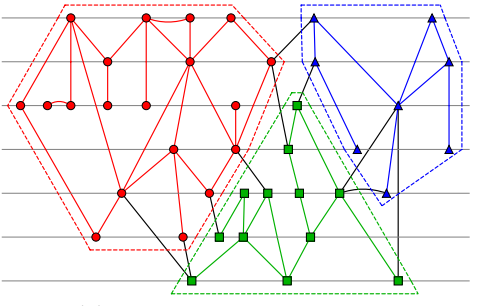
(a) Graph H_3 from [2], width optimized



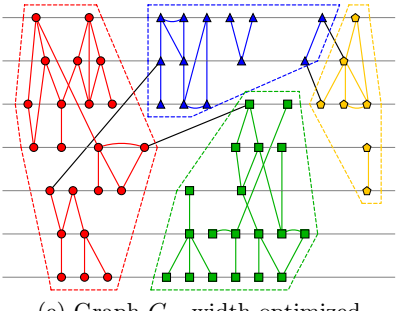
(b) Graph G_1, input



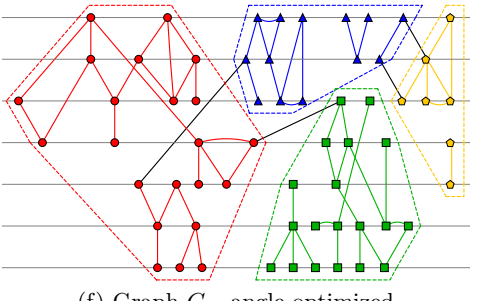
(c) Graph G_1, width optimized



(d) Graph G_1, angle optimized



(e) Graph G_2, width optimized



(f) Graph G_2, angle optimized

Fig. 10. Graph drawings produced by our LP formulation. Note that G_2 is not plane.