Straight-Line Drawing Algorithms for Hierarchical Graphs and Clustered Graphs*

Peter Eades † Qingwen Feng † Xuemin Lin ‡ Hiroshi Nagamochi §

November 25, 2004

Abstract

Hierarchical graphs and clustered graphs are useful non-classical graph models for structured relational information. Hierarchical graphs are graphs with layering structures; clustered graphs are graphs with recursive clustering structures. Both have applications in CASE tools, software visualization, and VLSI design. Drawing algorithms for hierarchical graphs have been well investigated. However, the problem of planar straight-line representation has not been solved completely. In this paper, we answer the question: does every planar hierarchical graph admit a planar straight-line hierarchical drawing? We present an algorithm that constructs such drawings in linear time. Also, we answer a basic question for clustered graphs, that is, does every planar clustered graph admit a planar straight-line drawing with clusters drawn as convex polygons? We provide a method for such drawings based on our algorithm for hierarchical graphs.

Keywords: Computational geometry, automatic graph drawing, hierarchical graph, clustered graph, straight-line drawing.

1 Introduction

A graph $G = (V, E)$ consists of a set V of vertices and a set E of edges, that is, pairs of vertices. Graphs are commonly used to model relations in computing, and many systems for manipulating graphs have recently been developed. Examples include CASE tools [55], knowledge representation systems [31], software visualization tools [54], and VLSI design systems [26]. A *graph drawing algorithm* reads as input a combinatorial description of a graph, and produces as output a visual representation of the graph. Such algorithms aim to produce drawings which are easy to read and easy to remember. Many graph drawing algorithms have been designed, analyzed, tested and used in visualization systems [2, 33].

With increasing complexity of the information that we want to visualize, we need more structure on top of the classical graph model. Several extended graph models have been proposed [5, 26, 35, 49, 50]. In this paper, we consider two such models:

*This work was supported by a research grant from the Australian Research Council and the subsidy from Kyoto University Foundation. An extended abstract of this paper was presented at the Symposium on Graph Drawing (GD'96), Berkeley, California, 1996.

†School of Information Technologies, University of Sydney, Madsen Building, F09, NSW 2006, Australia. Email: {peter, wfeng}@it.usyd.edu.au

‡School of Computer Science and Engineering, University of New South Wales, Sydney NSW 2052, Australia. Email: lxue@cse.unsw.edu.au

§Department of Information and Computer Sciences, Toyohashi University of Technology, Toyohashi, Aichi 441-8580, Japan. Email: naga@ics.tut.ac.jp

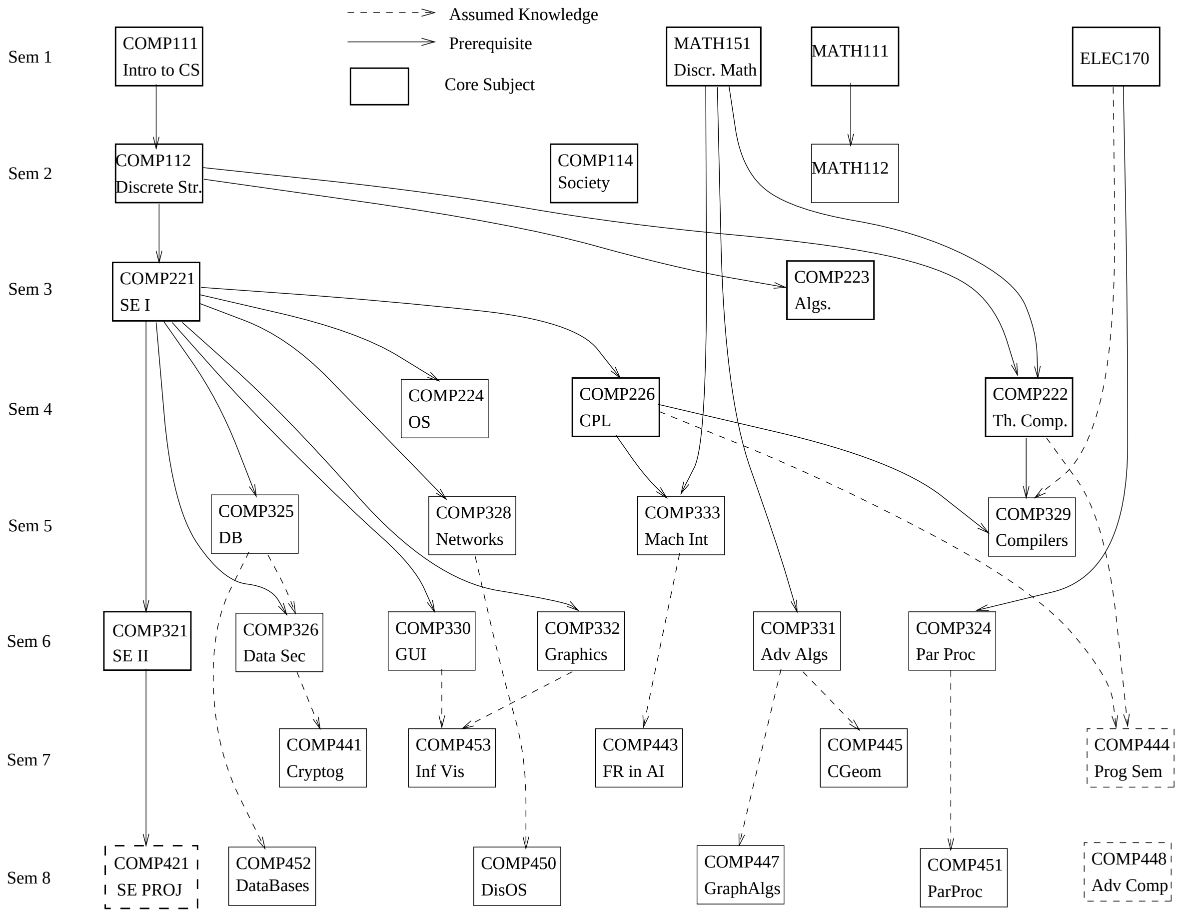


Figure 1: An example of a hierarchical graph.

- *Hierarchical graphs* are graphs with vertices assigned to layers. Hierarchical graphs appear in applications where hierarchical structures are involved [24, 50]. For example, Figure 1 shows a prerequisite diagram for subjects in a Bachelor's degree in Computer Science; an edge from a to b means that a is a prerequisite of b, and each layer represents a semester of study. In the example of Figure 1, the vertices are assigned to layers for semantic reasons. In other examples, the layer assignment is chosen to improve the readability of the drawing; see, for example, [11].

- *Clustered graphs* are graphs with recursive clustering structures which appear in many structured diagrams [26, 34, 35, 49]. For example, Figure 2 shows a relational diagram of some organizations in New South Wales; an edge between a and b indicates a joint project between a and b, and each cluster represents a group of organizations.

Both hierarchical graphs and clustered graphs have the power of representing certain additional structures required by applications. The drawings of hierarchical graphs and clustered graphs should reflect these structures, and therefore must meet additional constraints.

A graph $G = (V, E)$ is drawn by specifying a location in the plane for each vertex in V and a route (a simple Jordan curve) for each edge in E. The drawing is *planar* if no pair of edge

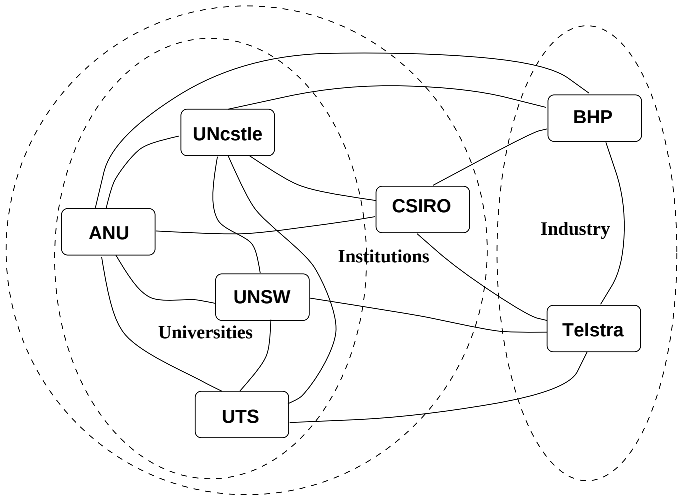


Figure 2: An example of a clustered graph.

routes cross, and the graph is planar if it admits a planar drawing. The planarity property has been the object of much of Graph Theory. For visualization purposes, it is well established that edge crossings significantly inhibit readability [40], and many algorithms for constructing planar drawings have been developed [2, 33].

A hierarchical graph is drawn with vertices of a layer on the same horizontal line, and edges as curves monotonic in y direction. A hierarchical graph is *hierarchical planar (h-planar)* if it admits a drawing without edge crossings. Algorithms for testing hierarchical planarity are presented in [29].

For a clustered graph, the clustering structure is represented by a closed curve that defines a region. The region contains the drawing of all the vertices which belong to that cluster. A clustered graph is *compound planar (c-planar)* if it admits a drawing with no edge crossings or edge-region crossings. Algorithms for testing compound planarity are presented in [25, 22, 20].

The hierarchical structure in hierarchical graphs imposes constraints on the y-coordinate, since all vertices of the same layer have to be drawn on exactly the same horizontal line. However, there are no constraints on the other dimension, that is, the x-coordinate. The clustering structure in clustered graphs can be viewed as constraints on both dimensions, that is, all vertices of the same cluster are restricted to a region.

One of the basic graph drawing conventions consists of representing edges as straight-line segments. The straight-line drawing convention is widely used in visualization. Graph drawing systems such as the Graph Layout Toolkit [46], GraphEd [28] and Diagram Server [3] contain a module for creating straight-line drawings of classical graphs. Intuitively, the eye can follow a straight-line easily; Sugiyama [48] and Batini [51] list straightness of lines as an important aim for graph drawing algorithms. This intuition has been confirmed by human experiments [41, 40]. Consequently, the straight-line drawing convention is one of many important modern aesthetic criteria [2] in graph drawing. More importantly, there are several general methods for drawing graphs which begin by adding dummy vertices on edges and then apply a straight-line drawing algorithm. This demand has spawned a considerable amount of attention to straight-line drawings in the research community.

For classical graphs, it is well known that every planar graph admits a straight-line drawing without edge crossings [18, 47, 53]. Tutte [52] proved that every triconnected planar graph admits a planar straight-line drawing where all the face boundaries are drawn as convex polygons. Algorithms for such drawings have also been investigated by Chiba et al. [8, 9]. More recently, efficient algorithms for planar straight-line drawings were developed by de Fraysseix et al. [23], Schnyder [45], Chrobak [10] and Kant [32]. These recent methods show that every planar graph admits a straight-line drawing in which each vertex is located at an integer grid point and the whole drawing uses $O(n^2)$ area.

In this paper, we investigate the problem of drawing both hierarchical graphs and clustered graphs with straight-line edges such that there are no crossings (edge crossings, edge-region crossings). The first question we address is of fundamental significance for drawing hierarchical graphs: does every planar hierarchical graph admit a planar straight-line hierarchical drawing? While many algorithms have been developed to draw hierarchical graphs [7, 11, 12, 24, 38, 43, 50], they all introduce bends to route the edges, and the basic problem of planar straight-line drawings has not been solved completely. It has been shown by Di Battista and Tamassia [4] that every planar st-graph admits an *upward drawing*, that is, a drawing where all arcs are drawn as straight-line segments pointing upward. However, the problem for hierarchical graphs is different, because we have more constraints: vertices of the same layer should be drawn on the same horizontal line and the layers should be an equal distance apart. A method to construct straight-line drawings of planar hierarchical graphs was presented by Eades, Lin and Tamassia [15]. They use a technique similar to that in [52] by Tutte, finding the position for every vertex in a global manner. In their algorithm, dummy vertices are added to transform an edge that spans more than two layers to a sequence of edges, each of which spans two consecutive layers. In fact, by the convexity of the drawing, the dummy vertices do not produce bends. The problem with the algorithm of [15] is that it only works for a special class of hierarchical graphs.

Here we present an algorithm that works for any planar hierarchical graphs. We use a divide and conquer approach, finding the position for every vertex in a recursive manner. The core of the algorithm is finding a suitable partition of the graph.

The second question addressed in this paper is for clustered graphs: does every planar clustered graph admit a planar straight-line drawing with clusters drawn as convex polygons? An algorithm for straight-line drawing of clustered graphs has been presented by Feng, Cohen, and Eades [20]. Again, however, it only applies to a special class of graphs with a certain strong connectivity property. In particular, it only applies to planar clustered graphs where every cluster induces a connected planar subgraph.

The rest of the paper is organized as follows. In section 2, we present some terminology for hierarchical graphs. In section 3, we prove that every planar hierarchical graph admits a planar straight-line drawing. A linear time algorithm that produces such drawings is presented. We introduce the clustered graph model in section 4. In section 5, we show that every planar clustered graph admits a planar straight-line convex cluster drawing. This is accomplished by transforming clustered graphs into hierarchical graphs. Based on this, we present an algorithm that computes such drawings in linear time in terms of the output size. In section 6, we discuss some examples, and pose some open problems.

2 Hierarchical Graphs

In this section we introduce the terminology, and some fundamental properties of hierarchical graphs.

A directed edge with a tail u and a head v is denoted by (u, v). A *hierarchical graph* $H = (V, A, \lambda, k)$ consists of a directed graph (V, A), a positive integer k, and, for each vertex u, an integer $\lambda(u) \in 1, 2, \ldots, k$, with the property that if $(u, v) \in A$, then $\lambda(u) < \lambda(v)$. For $1 \leq i \leq k$, the set $\{u \mid \lambda(u) = i\}$ is the ith *layer* of H and is denoted by L_i. The *span* of an edge (u, v) is $\lambda(v) - \lambda(u)$. An edge of span greater than one is *long*, and a hierarchical graph with no long edges is *proper*. For each vertex v in H, denote $\{u \in V \mid (v, u) \in A\}$ by $V_H^+(v)$ and $\{u \in V \mid (u, v) \in A\}$ by $V_H^-(v)$. A vertex v is called a *source* (respectively *sink*) if $V_H^-(v) = \emptyset$ (respectively $V_H^+(v) = \emptyset$). For a non-sink vertex v, a vertex $w \in V_H^+(v)$ is called an *up-neighbor* of v. Further, w is called the *highest up-neighbor* if $\lambda(w) = \max\{\lambda(u) \mid u \in V_H^+(v)\}$. Similarly, for a non-source vertex v, a vertex $w \in V_H^-(v)$ is called a *down-neighbor* of v, and w is called the *lowest down-neighbor* if $\lambda(w) = \min\{\lambda(u) \mid u \in V_H^-(v)\}$.

A *drawing* of a graph $G = (V, E)$ assigns a position $p(v) = (x(v), y(v))$ to each vertex $v \in V$ and a curve joining $p(u)$ and $p(v)$ to each edge $(u, v) \in E$. A hierarchical graph is conventionally drawn with layer L_i on the horizontal line $y = i$ (that is, $y(v) = \lambda(v)$ for all vertices v), and edges as curves monotonic in y direction. If no pair of non-incident edges intersect in the drawing, then we say it is a *hierarchical planar (h-planar)* drawing. Note that a non-proper hierarchical graph can be transformed into a proper hierarchical graph by adding dummy vertices on long edges. It is easily shown that a non-proper hierarchical graph is h-planar if and only if the corresponding proper hierarchical graph is h-planar. A *hierarchical planar embedding* of a proper hierarchical graph is defined by the ordering of vertices on each layer of the graph. Note that every such embedding has a unique external face. Also note that every proper h-planar graph admits a *straight-line hierarchical drawing*, that is, a drawing where edges are drawn as straight-line segments. However, for non-proper hierarchical graphs, the problem is not trivial, since no bends are allowed on long edges.

A *plane graph* refers to a planar graph embedded in the plane. In other words, a plane graph contains a planar graph and a planar embedding with a specified external face. We call a plane embedded hierarchical graph a *hierarchical plane graph.* If a hierarchical plane graph has only one source s and one sink t, then we call it a *hierarchical-st plane graph.* Observe that a hierarchical-st plane graph is a connected graph, and its source s and sink t must lie on the bottom layer and the top layer, respectively. In Section 3.2, we will show that every hierarchical plane graph can be extended to a hierarchical-st plane graph by adding $O(n)$ new vertices and edges.

The embedding of a hierarchical plane graph H determines, for every vertex v, a left-right relation among up-neighbors of v (see Figure 3). The head w of the rightmost (respectively leftmost) edge outgoing from v is called the *right up-neighbor* (respectively the *left up-neighbor*) of v, and is denoted by $r_H^+(v)$ (respectively $\ell_H^+(v)$). The *right down-neighbor* $r_H^-(v)$ and the *left down-neighbor* $\ell_H^-(v)$ of v are defined analogously.

Hierarchical graphs are directed graphs and thus we can borrow much of the standard terminology of graph theory [6]. The terms "path", "cycle", and "biconnectivity", when applied to a directed graph in this paper, refer to the underlying undirected graph. Note that we are interested in only *simple* cycles in this paper. To denote a cycle of a plane graph, we use the sequence of vertices on the cycle in clockwise order. For a cycle or path $\mathcal{P} = (v_1, v_2, \ldots, v_k)$, an edge between two non-consecutive vertices in $\mathcal{P}$ is called a *chord* of $\mathcal{P}$ (see Figure 4(a)). A cycle or path is called *chordless* if it has no chord. In hierarchical graphs, edges are directed from a lower layer to a higher layer. A path is called *monotonic* if the directions of the edges do not change along the path. In other words, a path is monotonic if the layer increases (or decreases) as we go along the path (see Figure 4(b)). Note that from a vertex v, a monotonic and chordless path from v to a sink can be obtained by traversing the highest up-neighbors one after another. Similarly, a monotonic and chordless path from a source to v can be found by tracing the lowest down-neighbors from v.

The following lemma gives some basic properties of the biconnected components of hierarchical-st plane graphs. Essentially it implies that, for hierarchical-st plane graphs, we can restrict our

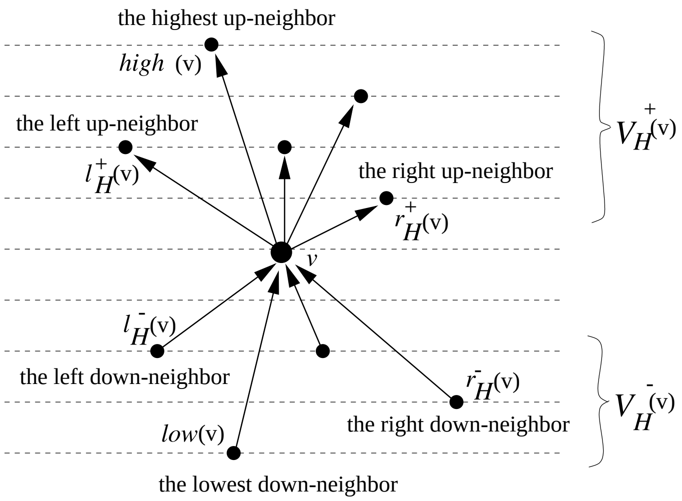


Figure 3: Definition of left-right relations in $V_H^-(v)$ and $V_H^+(v)$.

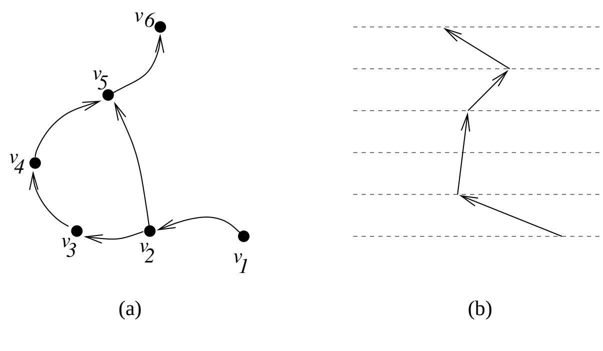


Figure 4: (a) A chord (v_2, v_5) in a path $(v_1, v_2, \ldots, v_6)$. (b) A monotonic path.

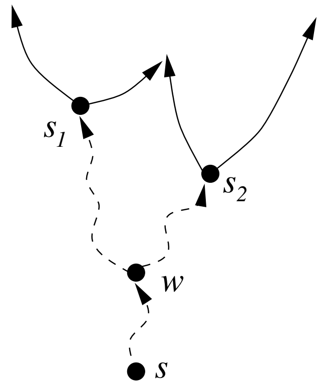


Figure 5: Illustration of the proof of Lemma 1-(a).

attention to the biconnected components.

Lemma 1 *Let $H = (V, A, \lambda, k)$ be a hierarchical-st plane graph, then:*

(a) *Every biconnected component of H is also a hierarchical-st plane graph.*

(b) *Suppose that B_1 and B_2 are two biconnected components of H, and for $i = 1, 2$, the source of B_i is s_i and the sink is t_i. Then* $\max(\lambda(s_1), \lambda(s_2)) \geq \min(\lambda(t_1), \lambda(t_2))$.

(c) *H has a planar straight-line hierarchical drawing if and only if each of its biconnected components has a planar straight-line hierarchical drawing.*

Proof: Let s and t be the source and sink of H, respectively.

(a) Let B be a biconnected component of H. The hierarchical planarity of B is inherited from H. Then it suffices to show that B has only one source and one sink. Suppose that B has two sources s_1 and s_2. Then H has two monotonic paths $\mathcal{P}_1$ from s to s_1 and $\mathcal{P}_2$ from s to s_2. Hence these paths have a common vertex w such that the sub-path of $\mathcal{P}_1$ from w to s_1 and the sub-path of $\mathcal{P}_2$ from w to s_2 are disjoint except for w (as in Figure 5). Consequently, all vertices on these two sub-paths belong to the same biconnected component B. This however contradicts the assumption that s_1 is a source of B. Similarly, we can show that B has a single sink.

(b) Since B_1 and B_2 are biconnected components of H, we can conclude from (a) that B_1 and B_2 are hierarchical-st plane graphs. Clearly, H has a monotonic path from s to s_1 and a monotonic path from s to s_2 in H. These give rise to a path $\mathcal{P}_s = (s_1, \ldots, s, \ldots, s_2)$ (not necessarily monotonic); and for every vertex w on $\mathcal{P}_s$, $\lambda(w) \leq \max\{\lambda(s_1), \lambda(s_2)\}$ holds. Similarly, H also has a path $\mathcal{P}_t = (t_1, \ldots, t, \ldots, t_2)$, and for every vertex z on $\mathcal{P}_t$, $\lambda(z) \geq \min\{\lambda(t_1), \lambda(t_2)\}$ holds. If $\max\{\lambda(s_1), \lambda(s_2)\} < \min\{\lambda(t_1), \lambda(t_2)\}$, then we see that $\mathcal{P}_s$ and $\mathcal{P}_t$ must be disjoint (see Figure 6). This, however, implies that B_1 and B_2 belong to the same biconnected component of H, contradicting our assumption. Therefore $\max\{\lambda(s_1), \lambda(s_2)\} \geq \min\{\lambda(t_1), \lambda(t_2)\}$.

(c) The only-if-part is immediate. We show the if-part. From the above (b), we see that the layers of every pair of biconnected components of H do not overlap except at the layer of their common vertex (that is, a cut vertex). Therefore, if each biconnected component of H has a planar straight-line hierarchical drawing, then we can construct a planar straight-line hierarchical drawing of H as a "chain" of the drawings of its biconnected components, as in Figure 7.

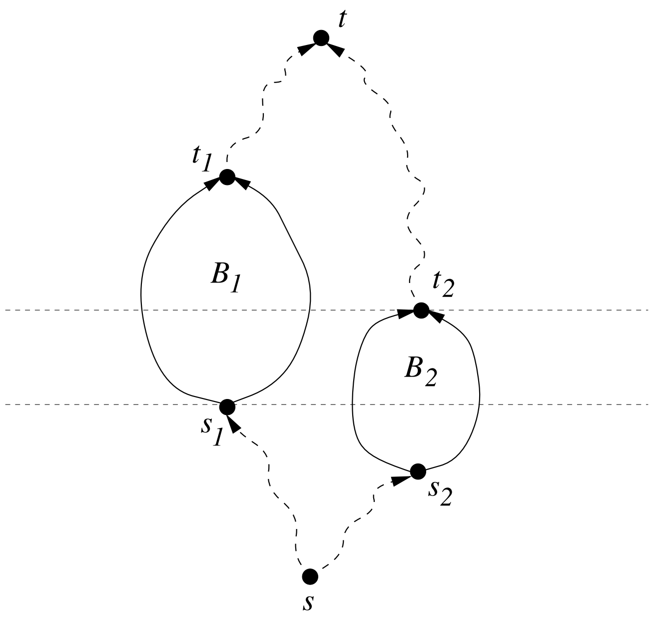


Figure 6: Illustration of the proof of Lemma 1-(b).

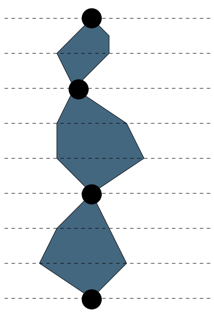


Figure 7: Drawing a chain of biconnected components.

□

From the above lemma, we can assume that a given hierarchical-st plane graph is biconnected, which implies that its external face is bounded by a simple cycle (except for the trivial case that the graph consists of a single edge or a single vertex).

In the next section, we present a method for dividing a biconnected hierarchical-st plane graph H into parts. The following lemma allows us to infer properties of the parts from the properties of H.

Lemma 2 *Let H be a hierarchical-st plane graph which is biconnected, and has the external facial cycle $\mathcal{C} = (v_1, \ldots, v_k, v_1)$. Suppose that $\mathcal{P} = (v_i, w_1, \ldots, w_l, v_j)$ is a monotonic path from v_i to v_j in H, and the vertices of $\mathcal{P}$ are not on $\mathcal{C}$ except v_i and v_j. Let H_1 and H_2 be the two subgraphs bounded inside by cycles $\mathcal{C}_1 = (v_1, \ldots, v_i, w_1, \ldots, w_l, v_j, \ldots, v_k, v_1)$ and $\mathcal{C}_2 = (v_i, \ldots, v_j, w_l, \ldots, w_1, v_i)$ inclusive. Then H_1 and H_2 are hierarchical-st plane graphs and are biconnected (see Figure 8).*

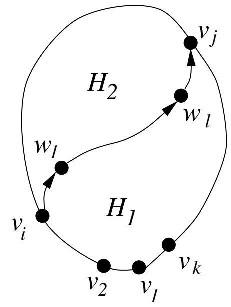


Figure 8: Illustration for Lemma 2.

Proof: The hierarchical planarity of H_1 and H_2 is immediate. Since $\mathcal{P} = (v_i, w_1, \ldots, w_l, v_j)$ is monotonic, both H_1 and H_2 have a single source and a single sink, respectively. The biconnectivity of H_1 and H_2 is immediate. □

3 Straight-Line Hierarchical Drawings

In this section, we show that given a hierarchical plane graph, a planar straight-line hierarchical drawing can be computed in linear time.

We apply a divide and conquer approach: divide the hierarchical graph into subgraphs, compute the drawings of the subgraphs, and obtain a drawing of the graph by combining the drawings of the subgraphs. The key part of this approach is to find a suitable partition.

For this purpose, we first assume that a given hierarchical plane graph satisfies two properties: (i) the boundary of the external face has a simple cycle and (ii) the boundary of every non-external face consists of exactly three edges. Such a hierarchical-st plane graph is called a *triangular hierarchical-st plane graph*.

Section 3.1 presents a proof, using the divide and conquer approach, that every triangular hierarchical-st plane graph admits a straight-line drawing. Based on the proof, Section 3.2 provides a straight-line drawing algorithm which runs in linear time. Further, in Section 3.2.1 we show that any hierarchical plane graph can be extended to a triangular hierarchical-st plane graph by adding $O(n)$ dummy vertices and edges in linear time.

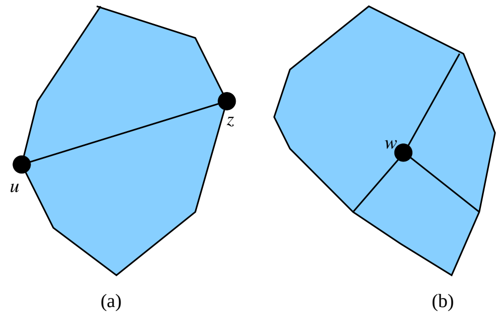


Figure 9: (a) Dividing into two parts along a chord, and (b) dividing into three parts.

3.1 Straight-line drawings of triangular hierarchical-st plane graphs

In this section we show that every triangular hierarchical-st plane graph admits a straight-line drawing with a prescribed convex polygon as its external face.

To describe the correspondence between the prescribed polygon and the external face, we need some terminology. An *apex* of a polygon is a geometric vertex of a convex polygon, that is, a vertex such that the two line segments incident to it form an angle not equal to 180 degrees. Let H be a hierarchical-st plane graph, let cycle $\mathcal{C}$ be the cycle of its external face. Note that there can be vertices of $\mathcal{C}$ which are not drawn as apexes of the polygon which forms the external face. If $\mathcal{C}$ has a chord (u, z) such that no vertex of $\mathcal{C}$ between u and z is at an apex, then a straight-line drawing would require that the edge (u, z) overlap with edges on $\mathcal{C}$. To help with this problem, we introduce some concepts. Let polygon P be a straight-line hierarchical drawing of cycle $\mathcal{C}$. We say that P is *feasible* for H if the following conditions hold:

- P is a convex polygon.
- If cycle $\mathcal{C}$ has a chord (u, z), then on each of the two paths of cycle $\mathcal{C}$ between u and z, there exists a vertex v which is drawn as an apex of polygon P.

We present a divide and conquer approach. For a given triangular hierarchical-st plane graph, we distinguish two situations, illustrated in Figure 9.

- If the external facial cycle has a chord (u, z), then we simply divide the graph into two parts with (u, z) in common. The input polygon is divided with a straight line between $p(u)$ and $p(z)$. Using Lemma 2, we apply recursion.
- If the external facial cycle has a no chord, then we find a vertex w not on the external facial cycle, such that there are three monotonic and chordless paths that connect w with the external facial cycle. We then divide the graph and the polygon into three parts; using Lemma 2 twice, we apply recursion.

The following lemma guarantees that such a partition always exists in a triangular hierarchical-st plane graph.

Lemma 3 *Let H be a triangular hierarchical-st plane graph with $n \geq 4$ vertices and with source s and sink t, and let $\mathcal{C}$ be the external facial cycle of H. Let $v(\neq s, t)$ be a vertex on the right path from s to t along $\mathcal{C}$. Then*

(i) *If both* $\ell^-_H(v)$ *and* $\ell^+_H(v)$ *are on* $\mathcal{C}$*, then one of* $(\ell^-_H(v), \ell^+_H(v))$*,* $(v, \ell^+_H(v))$ *and* $(\ell^-_H(v), v)$ *must be a chord on* $\mathcal{C}$*.*

(ii) *If a vertex* $w \in \{\ell^-_H(v), \ell^+_H(v)\}$ *is not on* $\mathcal{C}$*, then for some vertices* $u, z (\neq v)$ *on* $\mathcal{C}$*, there are two monotonic and chordless paths* $\mathcal{P}_{u,w} = (u, \dots, w)$ *and* $\mathcal{P}_{w,z} = (w, \dots, z)$*, where all vertices in* $\mathcal{P}_{u,w}$ *and* $\mathcal{P}_{w,z}$ *other than* u *and* z *are not on* $\mathcal{C}$*.*

Proof: Since H is triangulated, three vertices $v, \ell^-_H(v)$ and $\ell^+_H(v)$ form a triangle, where $\lambda(\ell^-_H(v)) < \lambda(v) < \lambda(\ell^+_H(v))$ holds.

(i) Assume that both $\ell^-_H(v)$ and $\ell^+_H(v)$ are on $\mathcal{C}$. Then it is immediate that either $(\ell^-_H(v), \ell^+_H(v))$ or $(v, \ell^+_H(v))$ or $(\ell^-_H(v), v)$ is a chord on $\mathcal{C}$.

(ii) Assume that a vertex $w \in \{\ell^-_H(v), \ell^+_H(v)\}$, say $w = \ell^+_H(v)$ is not on $\mathcal{C}$. To find a path $\mathcal{P}_{w,z} = (w, \dots, z)$ from w to a vertex z on $\mathcal{C}$, we traverse the highest up-neighbors from w until we come across a vertex z on $\mathcal{C}$. Clearly, $z \neq v$, and the path is monotonic and chordless. Similarly, we choose a path $\mathcal{P}_{u,w} = (u, \dots, w)$ by traversing the lowest down-neighbors from w until we reach a vertex u on $\mathcal{C}$. Clearly, $\mathcal{P}_{u,w}$ is monotonic and chordless. We see that $u \neq v$, because v is not the lowest down-neighbor of $w = \ell^+_H(v)$ by $\ell^-_H(v) \in V^-_H(w)$ and $\lambda(\ell^-_H(v)) < \lambda(v) < \lambda(\ell^+_H(v))$. The case that $\ell^-_H(v)$ is not on $\mathcal{C}$ can be treated analogously. □

Clearly, the above lemma holds for the case that v is on the left path from s to t along C, by replacing $\ell^-_H(v)$ and $\ell^+_H(v)$ in the statement with $r^-_H(v)$ and $r^+_H(v)$, respectively. Next we prove the main theorem.

Theorem 1 *Suppose that* H *is a triangular hierarchical-st plane graph, and polygon* P *is a straight-line hierarchical drawing of its external facial cycle* $\mathcal{C}$*. If* P *is feasible for* H*, then there exists a planar straight-line hierarchical drawing of* H *with external face* P*.*

Proof: We prove the claim by induction on the number n of vertices of H. The basis of the induction, $n = 3$ is immediate. Now, assume that the theorem holds for graphs with less than n vertices. Since P is convex, there is a vertex v other than the source s or sink t on the external face, such that v is drawn as an apex of P. We distinguish two cases:

Case 1: The external facial cycle $\mathcal{C}$ of H has a chord (u, z). By Lemma 2, chord (u, z) divides H into two subgraphs H_1 and H_2. We draw a straight line segment between $p(u)$ and $p(z)$, which divides P into two polygons P_1 and P_2. It can be verified that P_1 and P_2 are feasible for H_1 and H_2 as P is feasible and both u and z are apexes of P_1 and P_2. Since both H_1 and H_2 have less than n vertices, by induction, there exist straight-line hierarchical drawings of H_1 and H_2 with external faces P_1 and P_2. Hence, by combining the two drawings, we obtain a straight-line hierarchical drawing of H with external face P.

Case 2: The external facial cycle $\mathcal{C}$ of H has no chord. Suppose that v is a vertex such that $p(v)$ is an apex of P. By Lemma 3, there are vertices w, u, z and two monotonic and chordless paths $\mathcal{P}_{u,w} = (u, \dots, w)$ and $\mathcal{P}_{w,z} = (w, \dots, z)$ such that no internal vertex of either path is on $\mathcal{C}$. The inside of H has two monotonic paths $\mathcal{P}_1 = (u, \dots, w, \dots, z)$ (which consists of $\mathcal{P}_{u,w}$ and $\mathcal{P}_{w,z}$) and $\mathcal{P}_2$ consisting of a single edge between v and w. By applying Lemma 2, we can divide H into three parts (see Figure 10):

- $H_0(v)$ bounded by cycle $(u, \dots, z, \dots, w, \dots u)$;
- $H_1(v)$ bounded by cycle $(u, \dots, w, v, \dots u)$;
- $H_2(v)$ bounded by cycle $(z, \dots, v, w, \dots z)$.

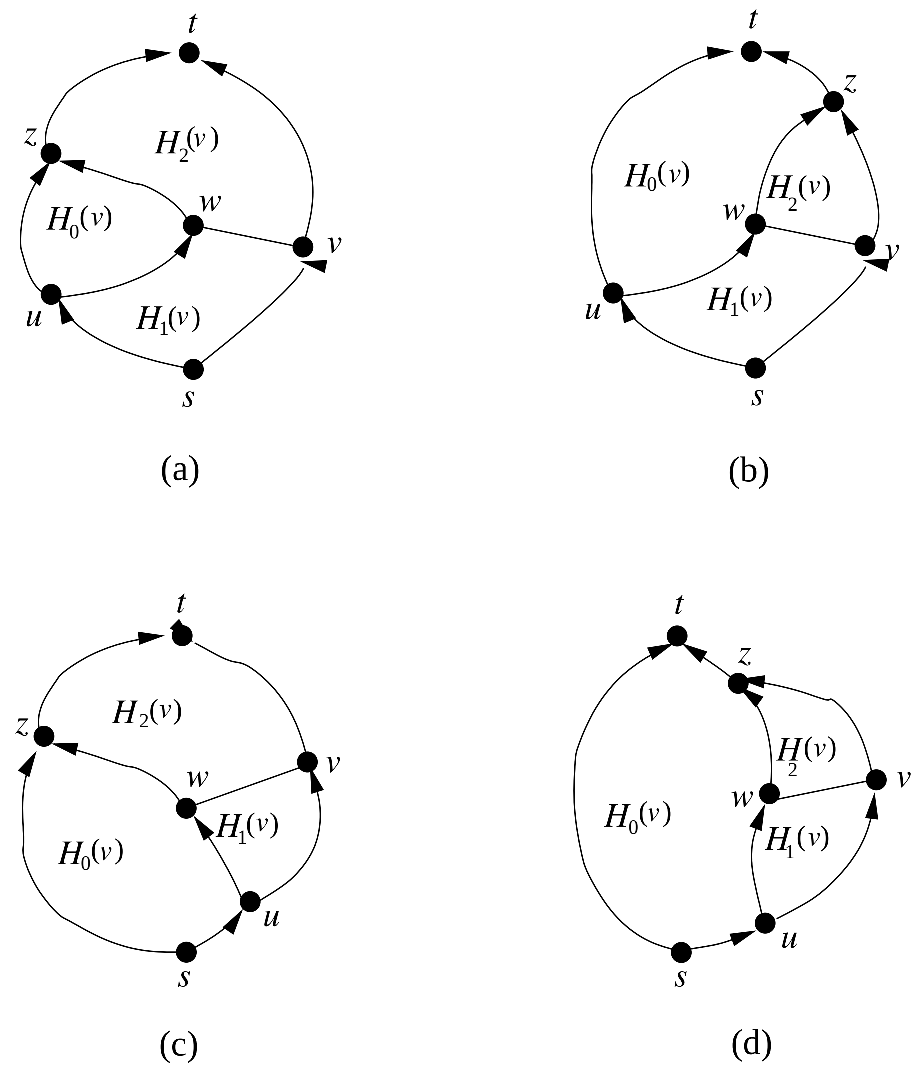


Figure 10: All possible partitions of H.

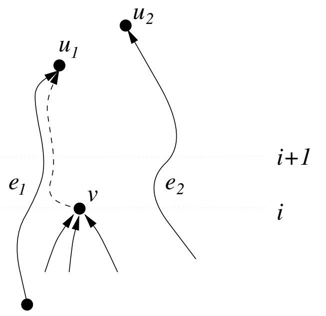


Figure 11: Eliminating a sink.

It is clear that $H_0(v)$, $H_1(v)$ and $H_2(v)$ are triangular hierarchical-st plane graphs.

Let $H_{frame}(v)$ be the graph that consists of only the external faces of $H_0(v)$, $H_1(v)$ and $H_2(v)$. Now $H_{frame}(v)$ has the same external face as H; hence polygon P is also a hierarchical planar drawing of the external face of $H_{frame}(v)$. We need a position $p(w)$ for w such that the drawings of the three internal faces of $H_{frame}(v)$ are convex polygons. For this, it is sufficient to draw vertex w strictly inside the triangle formed by the three positions $p(u)$, $p(v)$ and $p(z)$. The y coordinate $y(w) = \lambda(w)$ is fixed by the layering. We can always choose an appropriate x-coordinate $x(w)$, because $\lambda(u) < \lambda(w) < \lambda(z)$ and the three positions $p(u)$, $p(z)$ and $p(v)$ are not on a straight line (since v is chosen as an apex on the convex polygon). For instance, if $\lambda(w) < \lambda(v)$ then we can choose $x(w)$ anywhere in the range $x_a < x(w) < x_c$, where

$$x_a = \frac{\lambda(w) - \lambda(u)}{\lambda(z) - \lambda(u)} x(u) + \frac{\lambda(z) - \lambda(w)}{\lambda(z) - \lambda(u)} x(z),$$

$$x_c = \frac{\lambda(w) - \lambda(u)}{\lambda(v) - \lambda(u)} x(u) + \frac{\lambda(v) - \lambda(w)}{\lambda(v) - \lambda(u)} x(v).$$

We place other internal vertices of $H_{frame}(v)$ (which have degree 2, respectively, in $H_{frame}(v)$) onto the line segments from u to w and from w to z at appropriate horizontal lines according to their y-coordinates.

Since w is drawn strictly inside the triangle formed by positions $p(u)$, $p(v)$ and $p(z)$, the vertex w divides polygon P into three convex polygons P_0, P_1 and P_2, and vertices u, z, v and w are drawn as apexes of these polygons. Clearly, the source and sink of $H_0(v)$ (respectively $H_1(v)$ and $H_2(v)$) are at the smallest and greatest y-coordinates on P_0 (respectively P_1 and P_2), and they are therefore apexes of P_0 (respectively P_1 and P_2). Note that edges on path $(u, \ldots, w)$ are drawn on the same line, as are the edges on path $(w, \ldots, z)$, that is, the vertices on these paths are not apexes. However, since there are no chords on these paths, P_0, P_1 and P_2 are feasible for $H_0(v)$, $H_1(v)$ and $H_2(v)$, respectively.

As each of $H_0(v)$, $H_1(v)$ and $H_2(v)$ has less than n vertices, by induction there exist straight-line hierarchical drawings of $H_0(v)$, $H_1(v)$ and $H_2(v)$ with external faces P_0, P_1 and P_2. Hence, by combining these drawings, we obtain a straight-line hierarchical drawing of H with external face P. $\square$

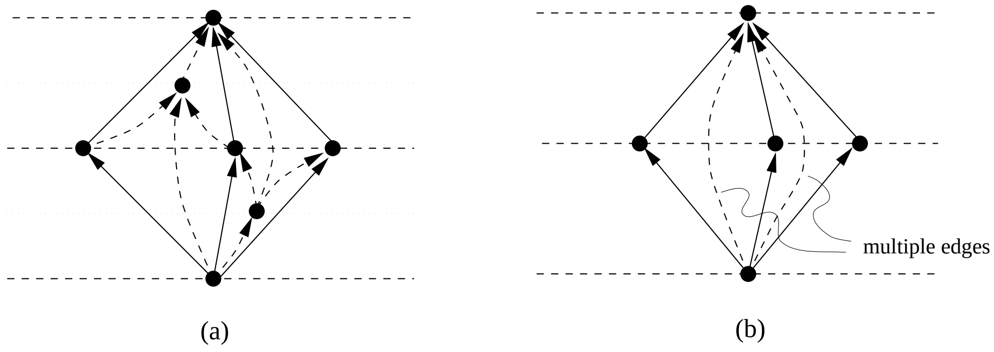


Figure 12: Triangulating the hierarchical graph.

3.2 The algorithm

The algorithm to compute a planar straight-line hierarchical drawing is based on the proof of Theorem 1. The input of the algorithm is a hierarchical plane graph H; the output is a planar straight-line hierarchical drawing of H. The algorithm consists of two phases: *Preprocessing* and *Drawing*. In the preprocessing phase, we extend the hierarchical plane graph to a set of triangular-st plane graphs. The drawing phase actually constructs a straight line drawing of each of those triangular-st plane graphs. Now we describe the two phases in more detail.

3.2.1 Preprocessing

We extend a given hierarchical plane graph to a set of triangular hierarchical-st plane graphs in four steps.

(1) Extend the hierarchical plane graph so that all the sources and sinks lie on the bottom layer and top layer. We can use a method similar to those in [4] and Chapter 2.2 of [39] which sweeps from bottom to top and from top to bottom to eliminate the sources and sinks in between (see Figure 11).

(2) Add a new vertex s below the bottom layer and connect it to all the sources; then add a new vertex t above the top layer and connect all the sinks to it. This gives a hierarchical-st plane graph.

(3) Compute all biconnected components of the hierarchical-st plane graph, each of which is a biconnected hierarchical-st plane graph by Lemma 1.

(4) Extend each of the biconnected hierarchical-st plane graphs obtained in (3) to a triangular one as follows: insert a layer between every two consecutive layers (this ensures that original layers are still evenly distributed); add a "star" structure inside each face (see Figure 12(a)), and place the center of each star on an inserted layer. After this, every internal face is bounded by exactly three edges. This triangulation method is a little unusual, but necessary. Figure 12(b) shows that if we do not add new vertices, multiple arcs can be produced. Further, we cannot allow dummy vertices on the arcs because this may introduce bends. Also note that no arcs are allowed between two vertices of the same layer.

Note that the size of the graph remains linear. Further, each of the four steps above can be carried out in linear time.

3.2.2 Drawing

The drawing phase is realized based on the proof of Theorem 1. Although the proof proceeds by a divide and conquer approach, we implement the drawing phase without using a recursive procedure, in order to obtain a linear time algorithm. The algorithm is described below. Here the drawing Γ is computed from the outside inward: at the start Γ consists of a drawing of the outside face, and the chords and paths described in Lemma 3 are added. Essentially the algorithm traverses the subgraphs H_i as described in the proof of Theorem 1; a queue $\mathcal{H}$ is used to manage this traversal.

Algorithm Hierarchical_Draw
Input: A triangular hierarchical-st plane graph H
Output: A planar straight-line hierarchical drawing Γ of H

(1) Initialize:

(1.1) Choose a convex polygon P_0 such that every vertex on the external facial cycle of H is an apex of P_0.

(1.2) Let $\Gamma := P_0$ and $\mathcal{H} := \{H\}$;

(1.3) If H is a triangle, then output Γ and halt.

(2) **While** $\mathcal{H} \neq \emptyset$ **do**

(2.1) Choose a subgraph $H' \in \mathcal{H}$ and an apex v in the polygon of the external face of H', and remove H' from $\mathcal{H}$.

(2.2) **If** there is a chord (u, z) stated in Lemma 3(i) for the apex v **then**:

(2.2.1) Divide H' into H'_1 and H'_2 with chord (u, z);

(2.2.2) Draw a straight-line segment between $p(u)$ and $p(z)$ in Γ, and let Γ be the resulting drawing;

(2.2.3) For $i = 1, 2$, if H'_i is not a triangle, then $\mathcal{H} := \mathcal{H} \cup \{H'_i\}$;

(2.3) **else**

(2.3.1) Find a vertex w adjacent to the apex v and paths $\mathcal{P}_{u,w} = (u, \ldots, w)$ and $\mathcal{P}_{w,z} = (w, \ldots, z)$ inside H', as stated in Lemma 3(ii);

(2.3.2) Divide H' into $H'_0 = H'_0(v)$, $H'_1 = H'_1(v)$ and $H'_2 = H'_2(v)$ with the edge e and paths $\mathcal{P}_{u,w}$, $\mathcal{P}_{w,z}$;

(2.3.3) Find a position $p(w)$ of w strictly within the three positions $p(u), p(v)$ and $p(z)$.

(2.3.4) Draw a straight-line segment from $p(w)$ to each of $p(u), p(v)$ and $p(z)$ in Γ;

(2.3.5) Let Γ be the resulting drawing, where the positions $p(w')$ of other vertices w' on $\mathcal{P}_{u,w}$ or $\mathcal{P}_{w,z}$ are determined by their layers;

(2.3.6) For $i = 0, 1, 2$, if H'_i is not a triangle, then $\mathcal{H} := \mathcal{H} \cup \{H'_i\}$;

(3) Output Γ and halt.

The correctness of this algorithm is immediate from Lemma 3 and the proof of Theorem 1. Figure 13 shows an example for the procedure.

We now present a linear time implementation of the above algorithm. We show that step (2.1) can be executed in $O(1)$ time, and both steps (2.2) and (2.3) can be executed in the time proportional to the number of edges in new straight line segments drawn in the step, that is, step (2.2) in $O(1)$ time and step (2.3) in $O(|\mathcal{P}_{u,w}| + |\mathcal{P}_{w,z}| + 1)$ time, where $|\mathcal{P}|$ denotes the number of edges in $\mathcal{P}$. It follows immediately that the total running time is $O(|A|) = O(n)$.

The linear time implementation requires some more terminology. We assume that for each $v \in V$, all neighbors in $V_H^+(v) = \{u_1, u_2, \dots, u_k\}$ are numbered by μ in increasing order from the leftmost neighbor $u_1 = \ell_H^+(v)$ to the rightmost $u_k = r_H^+(v)$, that is, $\mu(u_1) < \cdots < \mu(u_k)$, where μ is a mapping from V to the integer set. This ordering is given by the planar embedding of the graph and can be computed, for example, by the algorithm of Jünger and Mutzel [29]. We assume an analogous numbering on $V_H^-(v)$ for all $v \in V$. For a subgraph $H' \in \mathcal{H}$ of an input graph H, we denote the external facial cycle of H' by $\mathcal{C}(H')$, and the drawing of $\mathcal{C}(H')$ computed by the algorithm is denoted by $P(H')$.

A vertex v in a polygon P is *trivial* in P if v is the source or the sink of P or P is a polygon for a triangle $\mathcal{C}$. A nontrivial and non-apex vertex v in a polygon is a *pre-apex* of P. A nontrivial apex (respectively a pre-apex) v in P is a *right apex* (respectively *right pre-apex*) of P if v is on the right path from the bottom to the top in P. Similarly define *left apex* and *left pre-apex*.

During the execution of the algorithm, the apexes satisfy the following important properties.

1. If v is a right apex in (2.3), then every vertex $w'(\neq v, w, u, v)$ on path $\mathcal{P}_{u,w}$ (respectively $\mathcal{P}_{w,z}$) in step (2.3) becomes a right pre-apex of $P(H'_0(v))$ and a left pre-apex of $P(H'_1(v))$ (respectively $P(H'_2(v))$).

2. At any moment during the execution of the algorithm, each vertex v has at most one subgraph $H' \in \mathcal{H}$ such that v is a right apex or right pre-apex on $P(H')$.

3. Once a vertex v becomes a right apex of a polygon $P(H')$, there is always some subgraph $H'' \in \mathcal{H}$ such that v is a right apex of its polygon $P(H'')$ until v becomes a right apex of a triangle.

Similar results hold for a left apex v.

These properties suggest that we should maintain a set $AP(\mathcal{H})$ of apexes of subgraphs in $\mathcal{H}$ (that is, $AP(\mathcal{H}) = \{v \in V \mid v \text{ is an apex in } P(H') \text{ for some } H' \in \mathcal{H}\}$), instead of maintaining $\mathcal{H}$ explicitly. Further, for a subgraph H' and an apex v in (2.1), we need to maintain the neighbor sets $V_{H'}^-(v)$ and $V_{H'}^+(v)$ of v in such a way that steps (2.1), (2.2), and (2.3) can be executed efficiently. We now show how to do this.

For each $v \in V$, we define

$$R_{\mathcal{H}}(v) = [\ell_{H'}^-(v), r_{H'}^-(v), \ell_{H'}^+(v), r_{H'}^+(v)]$$

if there is a subgraph H' in the current $\mathcal{H}$ such that v is a right apex or right pre-apex on $P(H')$, and define $R_{\mathcal{H}}(v) = \emptyset$ otherwise. Similarly, let

$$L_{\mathcal{H}}(v) = [\ell_{H''}^-(v), r_{H''}^-(v), \ell_{H''}^+(v), r_{H''}^+(v)]$$

if there is a subgraph $H'' \in \mathcal{H}$ such that v is a left apex or left pre-apex on $P(H')$, and let $L_{\mathcal{H}}(v) = \emptyset$ otherwise. Also, let V_Γ and A_Γ be the sets of vertices and edges on the current drawing Γ, respectively.

In step (1), we can initialize $AP(\mathcal{H}) = \{v \in V \mid v \text{ is a nontrivial apex in } P_0\}$, V_Γ, A_Γ and $\{R_{\mathcal{H}}(v), L_{\mathcal{H}}(v)\}$, $v \in V$ in $O(n)$ time for $\mathcal{H} = \{H\}$.

We maintain $AP(\mathcal{H})$, $R_{\mathcal{H}}(v)$, and $L_{\mathcal{H}}(v)$ through each iteration of the while-loop (2). With this data then it is not difficult to see, from the proofs of Lemma 3 and Theorem 1, that we can determine whether such a chord (u, z) exists (in step (2.2)) in $O(1)$ time. If no such chord exists, then we can find a set $\{e, \mathcal{P}_{u,w}, \mathcal{P}_{w,z}\}$ in step (2.3) in $O(|\mathcal{P}_{u,w}|+|\mathcal{P}_{w,z}|+1)$ time; this can be done by maintaining the highest up-neighbor and lowest down-neighbor for each vertex. Note that finding the highest up-neighbor and lowest down-neighbor of each vertex for all vertices can be done in linear time. The highest up-neighbor and lowest down-neighbor of each vertex do not need to be updated as the algorithm proceeds; this is because that the algorithm needs the information only from the internal (not on external faces) vertices iteratively and the highest up-neighbor and lowest down-neighbor of each internal vertex in each iteration are never changed.

Provided that all the above data are properly updated after augmenting Γ in (2.2) or (2.3), we can check whether or not a new subgraph H'_i is a triangle in $O(1)$ time (and remove the corresponding trivial apex from $AP(\mathcal{H})$ if some H'_i is a triangle).

It is simple to update $AP(\mathcal{H})$ in constant time since every time at most four new apexes are added. Therefore, it suffices to show that for each $v \in V$, the data $\{R_{\mathcal{H}}(v), L_{\mathcal{H}}(v)\}$ can be updated in $O(1)$ time in (2.2) and (2.3).

First assume that a chord (u, z) is chosen in (2.2). In this case we need only to update $\{R_{\mathcal{H}}(u), L_{\mathcal{H}}(u)\}$ and $\{R_{\mathcal{H}}(z), L_{\mathcal{H}}(z)\}$, since the others remain unchanged in the new $\mathcal{H}$. Clearly, such an update can be done in $O(1)$ time as the only information involved in such an update is the original values of $\{R_{\mathcal{H}}(x), L_{\mathcal{H}}(x)\}$ ($x \in \{u, v, z\}$) and $\{u, v, z\}$. For instance, consider the case that $R_{\mathcal{H}}(z) \neq \emptyset$; that is, there is a H' in the current $\mathcal{H}$ such that z is a right apex or right pre-apex of $P(H')$. Suppose that the original $R_{\mathcal{H}}(z) = [\ell^-_{H'}(z), r^-_{H'}(z), \ell^+_{H'}(z), r^+_{H'}(z)]$. In this case, $R_{\mathcal{H}}(z)$ is updated to $[\ell^-_{H_1}(z), r^-_{H_1}(z), \ell^+_{H_1}(z), r^+_{H_1}(z)]$ where H' is divided to H_1 and H_2, and H_1 is on the left of (u, z). It can be immediately verified that $\ell^-_{H_1}(z) = \ell^-_{H'}(z)$, $r^-_{H_1}(z) = u$, $\ell^+_{H_1}(z) = \ell^+_{H'}(z)$, and $r^+_{H_1}(z) = r^+_{H'}(z)$. Note that if $R_{\mathcal{H}}(z) = \emptyset$ then it remains empty; this is because z won't become a right apex nor a right pre-apex after the partitioning.

Finally consider the case that no chord (u, z) exists, and an edge e and two paths $\mathcal{P}_{u,w}, \mathcal{P}_{w,z}$ is found in (2.3). In this case we can update $R_{\mathcal{H}}$ and $L_{\mathcal{H}}$ for the vertices $v, u, z \in V_\Gamma$ in a manner similar to (2.1). Also, when a vertex $v' \notin V_\Gamma$ becomes an apex (that is, $v' = w$) or a pre-apex (that is, $v'(\neq v, w, u, z)$ is on $\mathcal{P}_{u,w}$ or $\mathcal{P}_{w,z}$), we can easily compute an initial $\{R_{\mathcal{H}}(v'), L_{\mathcal{H}}(v')\}$ in $O(1)$ time if the right up-neighbor (left up-neighbor) and the right down-neighbor (left down-neighbor) of each vertex with respect to the original graph are computed and stored.

The above argument proves that algorithm Hierarchical_Draw runs in $O(n)$ time.

Theorem 2 *Let H be a hierarchical plane graph with n vertices. The above algorithm constructs a planar straight-line hierarchical drawing for H in $O(n)$ time and $O(n)$ space.*

Based on our results for hierarchical graphs, we next consider the straight-line drawing problem for clustered graphs.

4 Clustered Graphs

One of the fundamental questions in planar clustered graph drawing is: does every c-planar clustered graph admit a planar drawing such that edges are drawn as straight-line segments and clusters are drawn as convex polygons? In this section, we answer this question based on our results for hierarchical graphs. We transform a clustered graph into a hierarchical graph, and construct a straight-line convex cluster drawing on top of the straight-line hierarchical drawing.

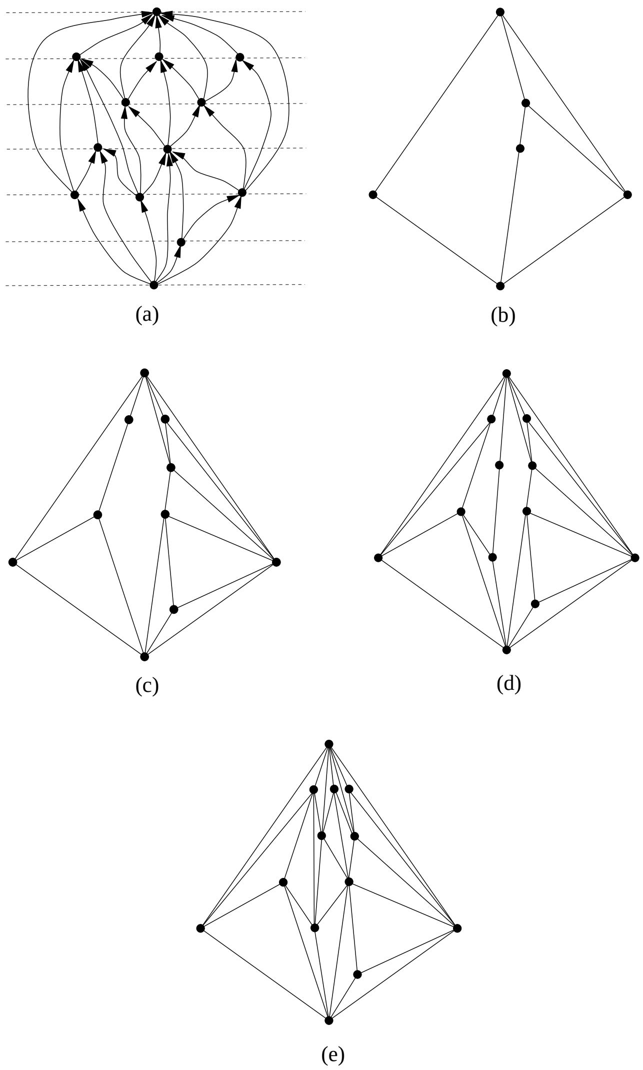


Figure 13: An Example: (a) A triangular hierarchical-st plane graph. (b)-(d) Intermediate drawings produced by the procedure. (e) The final drawing.

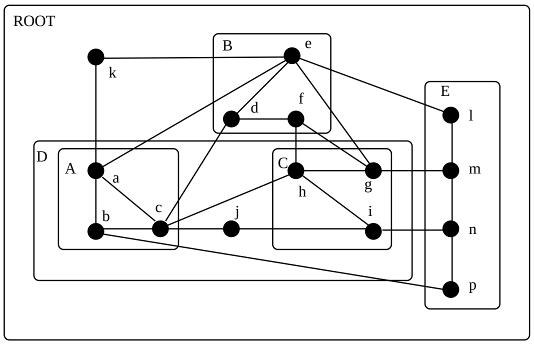


Figure 14: A c-planar clustered graph.

First, in section 4.1, we introduce some of the necessary notation, define c-planarity precisely, and state the relevant properties of c-planarity. Then, in section 4.2, we show how to construct an "c-st" numbering of the vertices; this is the most critical and most difficult part of the algorithm. The algorithm for constructing the c-st numbering runs in linear time. Using this ordering, we give in section 4.3 a transformation from clustered graphs to hierarchical graphs so that the algorithm of section 3.2 can be applied to produce c-planar drawings with convex clusters. The time complexity of the algorithm is linear in the output size.

4.1 Preliminaries

A *clustered graph* $C = (G, T)$ consists of an undirected graph $G = (V, A)$ and a rooted tree $T = (\mathcal{V}, \mathcal{A})$ such that the leaves of T are exactly the vertices of G. For a vertex $\nu \in \mathcal{V}$, let $chl(\nu)$ denote the set of children of ν, and $pa(\nu)$ denote the parent of ν (if ν is not a root). Each node ν of T represents a *cluster* $V(\nu)$, a subset of the vertices of G that are leaves of the subtree rooted at ν. Let $G(\nu)$ denote the subgraph of G induced by $V(\nu)$. Note that the tree T describes an inclusion relation between clusters. If a node ν' is a descendant of a node ν in the tree T, then we say that the cluster of ν' is a *sub-cluster* of ν. In a *drawing* of a clustered graph $C = (G, T)$, graph G is drawn as points and curves as usual. For each node ν of T, the cluster is drawn as a simple closed region R that contains the drawing of $G(\nu)$, such that:

- the regions for all sub-clusters of ν are completely contained in the interior of R;
- the regions for all other clusters are completely contained in the exterior of R;
- if there is an edge e between two vertices of $V(\nu)$, then the drawing of e is completely contained in R.

We say that the drawing of edge e and region R have an *edge-region crossing* if the drawing of e crosses the boundary of R more than once. A drawing of a clustered graph is *c-planar* if there are no edge crossings or edge-region crossings. If a clustered graph C has a c-planar drawing then we say that it is *c-planar* (see Figure 14).

A clustered graph $C = (G, T)$ is a *connected clustered graph* if each cluster $V(\nu)$ induces a connected subgraph $G(\nu)$ of G. The following results from Fengs thesis, [21] characterize c-planarity in a way which can be exploited by our drawing algorithm.

Theorem 3 *A connected clustered graph $C = (G,T)$ is c-planar if and only if graph G is planar and there exists a planar drawing of G, such that for each node ν of T, all the vertices and edges of $G - G(\nu)$ are in the external face of the drawing of $G(\nu)$.*

Let $C_1 = (G_1,T_1)$ and $C_2 = (G_2,T_2)$ be two clustered graphs such that T_1 is a subtree of T_2 and for each node ν of T_1, $G_1(\nu)$ is a subgraph of $G_2(\nu)$. We say that C_1 is a *sub-clustered-graph* of C_2.

Theorem 4 *A clustered graph $C = (G,T)$ is c-planar if and only if it is a sub-clustered graph of a connected and c-planar clustered graph.*

A c-planar embedding of a connected clustered graph can be found efficiently [21, 36]. In the rest of the paper, we assume that $C = (G,T)$ is a c-planar and connected clustered graph which has no degenerated clusters, that is, every non-leaf node of T has at least two children. Therefore the input size of $T = (\mathcal{V},\mathcal{A})$ is $O(|\mathcal{V}| + |\mathcal{A}|) = O(n)$, where n is the number of vertices in G.

The algorithm of the next section uses some tree operations. For two distinct nodes u and v in a rooted tree T, let $LCA(u,v)$ denote the *least common ancestor* of u and v in T, and $GUA(u,v)$ denote the pair (u',v') of the *greatest uncommon ancestors* (that is, u' (respectively v') is an ancestor of u (respectively v) that is a child of $LCA(u,v)$). By using the fast least common ancestor algorithm [27, 44], each query of finding $LCA(u,v)$ can be answered in $O(1)$ time after $O(n)$ time preprocessing. With a slight modification in the step 3 of the algorithm [44], $GUA(u,v)$ can be found in $O(1)$ time based on the same preprocessing.

In the following sections, we also assume that, for a given c-planar and connected clustered graph $C = (G,T)$, each face (including the external face) of G is a triangle. We triangulate G so that resulting clustered graph remains c-planar. This is accomplished by using a triangulation algorithm [30, 42] or by triangulating each face f introducing a new vertex v_f (together with edges between v_f and vertices on the cycle C_f of f), where $\{v_f\}$ will be a child cluster of the smallest cluster ν_f that contains the face f. The latter triangulation can be done in $O(n)$ time, because for each face f, the cluster ν_f can be computed in $O(|C_f|)$ time by using the the least common ancestor algorithm [27, 44].

4.2 The c-st numbering algorithm

In this section, we define the concept of "c-st numbering" and show how to compute it in linear time.

By Theorem 4, we assume that we are given a c-planar connected clustered graph $C = (G,T)$ with a c-planar embedding. For each vertex u, let $A(u)$ be a doubly-linked list of edges around u, where the edges in $A(u)$ appear along u in the order of the list in the embedding.

The *st numbering* of the vertices of a graph has proved to be a useful tool for many graph algorithms, especially graph drawing algorithms (see, for example, [16, 2]). We next review this concept. Suppose that (s,t) is an edge of a biconnected graph G with n vertices. In an *st numbering*, the vertices of G are numbered from 1 to n so that vertex s receives number 1, vertex t receives number n, and any vertex except s and t is adjacent both to a lower-numbered vertex and a higher-numbered vertex. Vertices s and t are called the *source* and the *sink* respectively. An *st* numbering of a biconnected graph can be computed in linear time [17].

An outline of our algorithm is described as follows. We need to generalize this notion to clustered graphs. Given a clustered graph $C = (G,T)$, an *st* numbering of the vertices of G such that the vertices that belong to the same cluster are numbered consecutively is a *c-st numbering*.

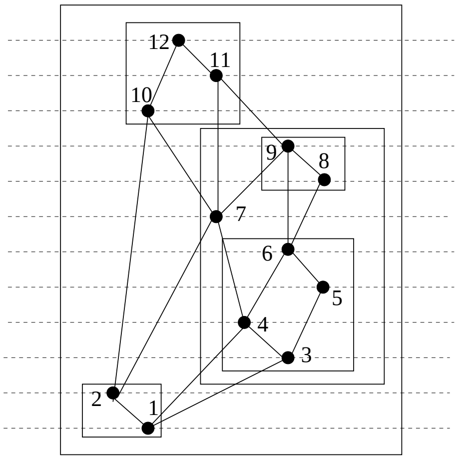


Figure 15: Clustered Graph → Hierarchical Graph

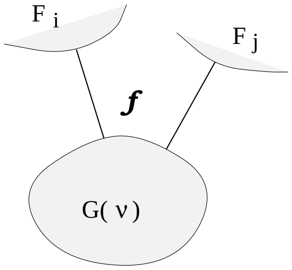


Figure 16: Illustration for the proof of Lemma 4.

Looking ahead to Section 4.3, the c-st numbering gives us a layer assignment of the vertices of G. Hence, the clustered graph is transformed to a hierarchical graph (see Figure 15), and each cluster has consecutive layers. Because of this property, we can show that a straight-line convex cluster drawing can be constructed from the straight-line hierarchical drawing.

The critical part of this method is the construction of the *c-st* numbering. The remainder of this section is devoted to the construction of the c-st numbering.

We construct some auxiliary graphs to compute such a *c-st* numbering. Note that *st* numberings are constructed on biconnected graphs. We need the following lemma to ensure appropriate connectivity of our auxiliary plane graphs.

Lemma 4 *Suppose that $C = (G, T)$ is a connected c-planar clustered graph, and G is triangulated. Then, for every non-root node ν of T, the subgraph of G induced by $V - V(\nu)$ is connected.*

Proof: Suppose that the subgraph of G induced by $V - V(\nu)$ has k components, denoted by $F_1, \ldots, F_k$, $k \geq 1$. Hence, there are no edges that connect vertices of F_i to vertices of F_j ($1 \leq i, j \leq k$, $i \neq j$).

Since G is triangulated, it has a unique planar embedding. By Theorem 3, all vertices and edges of $G - G(\nu)$ are in the same face of $G(\nu)$. Hence, all edges that connect $G(\nu)$ and F_i ($i = 1, \ldots, k$)

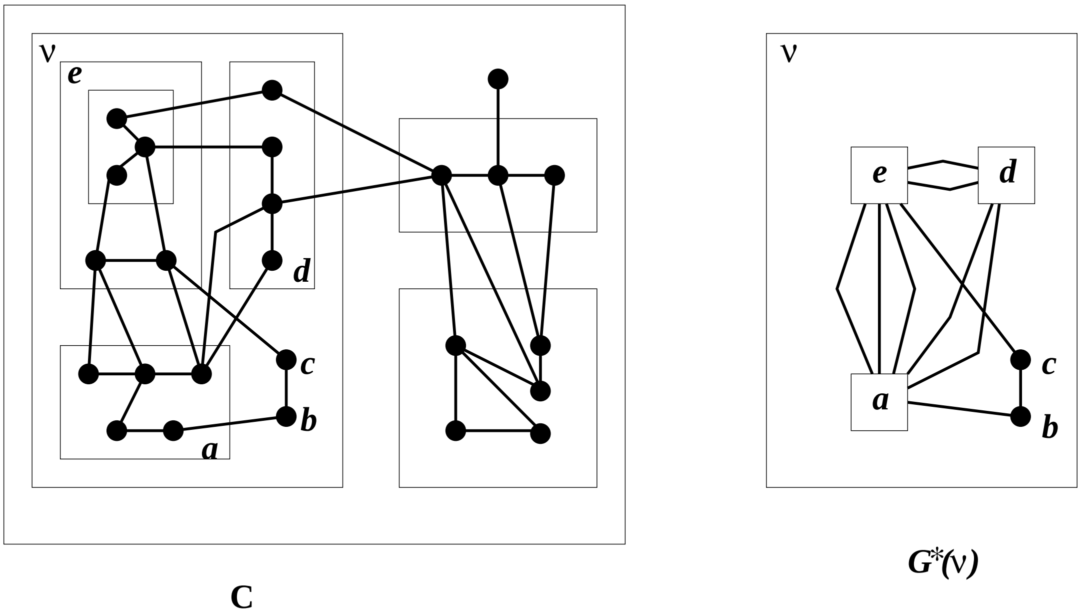


Figure 17: A clustered graph C with a node ν; and the graph $G^*(\nu)$.

are in the same face of $G(\nu)$. Suppose that $k \geq 2$; then there is a face f of G whose boundary contains an edge that connects $G(\nu)$ and F_i, and also an edge that connects $G(\nu)$ and F_j, and $i \neq j$ (see Figure 16). Because G is triangulated, the face f is bounded by exactly three edges. Therefore, the boundary of the face f also contains an edge that connects F_i and F_j. This contradicts the fact that there are no edges that connect vertices of F_i and vertices of F_j ($1 \leq i, j \leq k$, $i \neq j$). We deduce that $k = 1$. □

The critical property of a *c-st* numbering is that the vertices of the same cluster are numbered consecutively. To compute a numbering with this property, we proceed down the tree from the root; at each non-leaf cluster ν we order the child clusters of ν. This involves the computation of an auxiliary graph $G^*(\nu)$, as follows. Intuitively, $G^*(\nu)$ is the graph obtained from $G(\nu)$ by shrinking each child cluster $V(\mu)$ ($\mu \in chl(\nu)$) into a single vertex; it may contain multiple edges. An example is shown in Figure 17. More precisely, the vertex set of $G^*(\nu)$ is $chl(\nu)$, and the edge set $E(\nu)$ of $G^*(\nu)$, is $\{(u, v) \mid LCA(u, v) = \nu\}$. Clearly, each edge in E appears in exactly one of these graphs $G^*(\nu)$. Hence the total size of all graphs $G^*(\nu)$ is $\sum_{\nu \in \mathcal{V}}(|chl(\nu)| + |E(\nu)|) = O(|\mathcal{V}| + |E|) = O(n)$. Next we show how to compute $G^*(\nu)$ efficiently.

Lemma 5 *All graphs $G^*(\nu)$, $\nu \in \mathcal{V}$ can be computed in $O(n)$ time.*

Proof: We first partition the edge set E into the edge sets $E(\nu) = \{(u, v) \mid LCA(u, v) = \nu\}$, $\nu \in \mathcal{V}$. This partition can be computed in $O(n)$ time by computing all $LCA(u, v)$, $(u, v) \in E$ in $O(1)$ time per edge (u, v). To construct $G^*(\nu)$ explicitly, we however need to identify end vertices of each edge (u, v) in $E(\nu)$ by the names of $chl(\nu)$ (that is, $GUA(u, v)$). In other words, for each $\mu \in chl(\nu)$, we need to compute the adjacency list $A(\mu)$ of edges incident to μ. We represent $A(\mu)$ as a doubly linked list, which is initially empty. Then we repeatedly choose an edge (u, v) from $E(\nu)$, compute $GUA(u, v) = (\mu, \mu')$, and add the edge (u, v) to $A(\mu)$ and $A(\mu')$. The resulting is $G^*(\nu)$. It is easy to see that the above procedure for constructing a $G^*(\nu)$ takes $O(|E(\nu)|)$ time (except the $O(n)$

time preprocessing in the least common ancestor algorithm). This proves that all the graphs $G^*(\nu)$ can be computed in $O(|E|) = O(n)$ time. □

From this lemma, we can assume that all the graphs $G^*(\nu)$, $\nu \in \mathcal{V}$ are at hand. For later processing, it is important that for each node μ of $G^*(\nu)$, the set of edges incident to μ is stored in a doubly-linked list $A(\mu)$.

Using the graphs $G^*(\nu)$, we can compute a *c-st* numbering. The algorithm uses another auxiliary graph $F(\nu)$, derived from $G^*(\nu)$. We proceed from the root to the leaves in T during which a particular order of the children in $chl(\nu)$ is determined after each $F(\nu)$ is computed. The computation is slightly different when ν is the root. We describe this case first, and then the general case when ν is not the root.

Suppose that γ is the root of T; let the graph $F(\gamma)$ be $G^*(\gamma)$. We now describe the computation of the *c-st* numbering for $F(\gamma)$. Firstly, we choose an edge (s, t) in the external facial cycle, of G such that $LCA(s, t) = \gamma$ (that is, such that (s, t) that does not "belong" to any other cluster except the root cluster.) Since the input clustered graph is connected, such an edge exists. Lemma 4 implies that deleting any node from $F(\gamma)$ does not increase the number of components. Therefore $F(\gamma)$ is biconnected, and hence we can compute an *st* numbering for a given source and a sink in $F(\gamma)$. By choosing the vertex $\mu \in chl(\gamma)$ with $s \in V(\mu)$ as the source, and the vertex $\mu' \in chl(\gamma)$ with $t \in V(\mu')$ as the sink, we compute an *st* numbering in $F(\gamma)$, and order children of γ according to this numbering.

Now suppose that $\nu \in \mathcal{V}$ is a non-root node. We can assume by induction that for any proper ancestor $\hat{\nu}$ of ν, the order of the children of $\hat{\nu}$ has already been determined by an *st* numbering in $F(\hat{\nu})$.

The graph $F(\nu)$ depends on the ordering of the children of its ancestors. It is constructed from $G^*(\nu)$ by adding two new vertices S and T and some edges between $G^*(\nu)$ and $\{S, T\}$ defined as follows.

- For each node $\mu \in chl(\nu)$ (that is, for each vertex in $G^*(\nu)$), we connect μ and S with a new edge if there is an edge $e = (u_1, u_2) \in E$ with $u_2 \in V(\mu)$ such that an ancestor ν_1 of u_1 is ordered before an ancestor ν_2 of u_2 among the children of $LCA(u_1, u_2)$ in T (hence $(\nu_1, \nu_2) = GUA(u_1, u_2)$). See Figure 18(a).

- We connect μ and T with a new edge if there is an edge $e = (u_1, u_2) \in E$ with $u_1 \in V(\mu)$ such that ν_1 is ordered before ν_2 for $(\nu_1, \nu_2) = GUA(u_1, u_2)$.

- We connect S and T with a new edge.

This forms graph $F(\nu)$ (see Figure 19). In the case that the vertex s belongs to the cluster ν, we simply choose the vertex which represents the child cluster that contains s as S; similarly for vertex t and vertex T.

Note that the graph $F(\nu)$ is not significantly larger than $G^*(\nu)$. Let $chl_S(\nu) \subseteq chl(\nu)$ (respectively $chl_T(\nu) \subseteq chl(\nu)$) denote the set of vertices that are adjacent to S (respectively T) in $F(\nu)$. Clearly each $F(\nu)$ has $|chl(\nu)| + 2$ vertices and $|chl_S(\nu)| + |chl_T(\nu)| + |E(\nu)|$ ($\leq 2|chl(\nu)| + |E(\nu)|$) edges.

Again, by using Lemma 4, one can see that $F(\nu)$ is biconnected. We order every vertex of $F(\nu)$ by computing an *st* numbering, choosing vertex S as the source, and vertex T as the sink. Since an *st* numbering can be constructed in linear time [17], an *st* numbering in $F(\nu)$ can be computed in $O(|chl(\nu)| + |E(\nu)|)$ time (if $F(\nu)$ is at hand) (Note that, if necessary, we can avoid multiple edges in $G^*(\nu)$ in the computation of the *st* numbering by inserting a dummy vertex on every edge.)

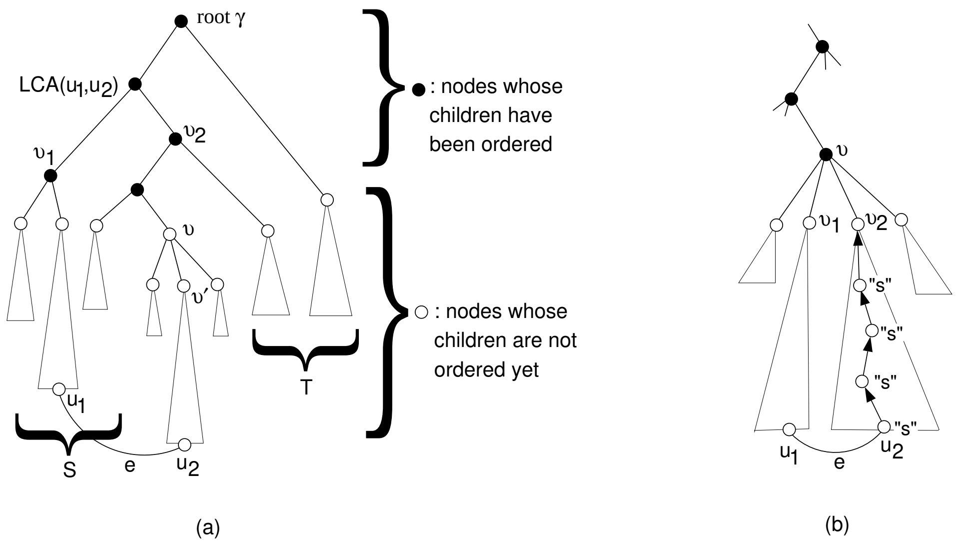


Figure 18: Illustration of computing graph $F(\nu)$.

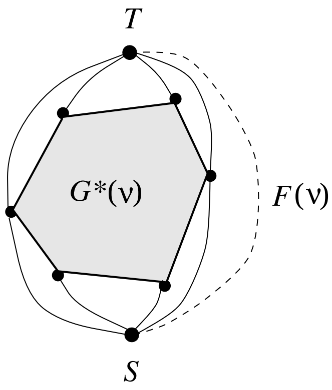


Figure 19: Illustration of graph $F(\nu)$.

It should be noted that the total number of edges of G between $V(\nu)$ and $V - V(\nu)$ over all $\nu \in \mathcal{V}$ may be $\Omega(n^2)$, although the total number of edges in all graphs $F(\nu)$, $\nu \in \mathcal{V}$ is $O(n)$. Thus, to compute all $F(\nu)$ in $O(n)$ time, we need to identify $chl_S(\nu)$ and $chl_T(\nu)$ without explicitly computing those edges between $V(\nu)$ and $V - V(\nu)$ in G. Somewhat surprisingly, this can be done.

Lemma 6 *All graphs $F(\nu)$, $\nu \in \mathcal{V}$ can be computed in $O(n)$ time.*

Proof: Clearly, for the root γ of T, $F(\gamma) = G^*(\gamma)$ and its *st*-numbering can be computed in $O(n)$ time. In what follows, we compute $F(\nu)$ for each node ν from the root to the leaves (visiting siblings in an arbitrary order), and show how to identify $chl_S(\nu)$ for each $\nu \in \mathcal{V}$; computing $chl_T(\nu)$ can be treated analogously. When the graph $F(\nu)$ for a node ν is computed, graphs $F(\hat{\nu})$ for all ancestors $\hat{\nu}$ of ν have been determined and hence all nodes which belong to $chl_S(\nu)$ in $G^*(\nu)$ also have been determined. That is, $chl_S(\nu)$ is determined by the set of nodes $\mu \in chl(\nu)$ such that there is an edge $e = (u_1, u_2) \in E$ with $u_1 \notin V(\mu)$ and $u_2 \in V(\mu)$, and ν_1 is ordered before ν_2 for $(\nu_1, \nu_2) = GUA(u_1, u_2)$ in the order of children of $LCA(u_1, u_2)$ (see Figure 18(a)). To identify such nodes efficiently, we perform the following operation for each edge (u_1, u_2) in $G^*(\nu)$ after constructing $F(\nu)$. Suppose that $u_1 \in V(\nu_1)$ and $u_2 \in V(\nu_2)$ for $\nu_1, \nu_2 \in chl(\nu)$, and assume that ν_1 is ordered before ν_2 in the *st* numbering of $F(\nu)$. Then, we mark with "*s*" all nodes in the path P_{u_2,ν_2} from the leaf node u_2 to ν_2 in T (see Figure 18(b)). During the traversal of P_{u_2,ν_2}, we can stop marking nodes once we encounter a node ν^* which is already marked "*s*", because we see by induction that in this case the rest of nodes from ν^* to ν_2 in P_{u_2,ν_2} have been marked "*s*". After applying this procedure to all edges in $G^*(\nu)$ and $G^*(\hat{\nu})$ for all ancestors $\hat{\nu}$ of ν, the desired $chl_S(\mu)$ for each $\mu \in chl(\nu)$ is given by the set of nodes in $chl(\mu)$ that have received mark "*s*". Thus any node will never be marked with "*s*" more than once, and it follows that the total time for marking operations is $O(|\mathcal{V}| + |E|)$. By applying the above procedure to each node from top to bottom in T, we can identify all $chl_S(\nu)$ in $O(n)$ time. Similarly, all $chl_T(\nu)$ can be obtained in $O(n)$ time. As mentioned above, all $F(\nu)$ can be computed in $O(n)$ time from $chl_S(\nu)$ and $chl_T(\nu)$. $\square$

After computing all the graphs $F(\nu)$, each cluster ν is assigned a number given by the order within the graph $F(pa(\nu))$. Therefore, a recursive hierarchy of orders is formed. We expand it lexicographically into a linear order and hence form an ordering of all vertices of G. It can be verified that this order gives us a *c-st* numbering, that is, an *st* numbering on the vertices of G such that the vertices that belong to the same cluster are numbered consecutively.

Lemma 7 *A* c-st *numbering of a triangulated and c-planar connected clustered graph $C = (G, T)$ can be computed in $O(n)$ time.*

4.3 The drawing algorithm for clustered graphs

Using the *c-st* numbering computed in the previous section, we transform a clustered graph into a hierarchical graph by assigning the layer of each vertex with its *c-st* number. Then we apply the straight-line hierarchical drawing algorithm described in section 3, and obtain a planar straight-line hierarchical drawing of G. The *c-st* numbering ensures that each cluster occupies consecutive layers in the drawing. For every cluster, we draw a convex hull of the vertices of the cluster. Clearly, the convex hull of a cluster contains the convex hulls of its sub-clusters. The *c-st* numbers within a cluster are consecutive, thus if the convex hulls of two clusters overlap in y-coordinate, then one is a sub-cluster of the other. This keeps the clusters apart; for example, if vertex u lies inside the convex hull for cluster ν, then u is a member of $V(\nu)$. In general, if a cluster ν is not a sub-cluster of ν' and neither is cluster ν' a sub-cluster of ν, then the convex hulls of ν and ν' are disjoint.

By Theorem 2, there are no edge crossings in the drawing, since our hierarchical planar drawing algorithm does not produce any edge crossings.

Since we are given a connected clustered graph, each cluster forms a connected subgraph of G. If ℓ is a straight-line segment with endpoints outside the convex hull for cluster ν, and ℓ intersects the convex hull for ν, then ℓ crosses an edge in $G(\nu)$. Therefore there are no edges that cross the region (the convex hull) of a cluster where they do not belong, because otherwise there would be an edge crossing.

A convex hull of a given simple polygon with m apexes can be constructed in $O(m)$ time [39]. In fact, there is a simple $O(m)$ time algorithm for computing a convex hull of a set of m points which are already sorted by their y-coordinates. Since all vertices in each cluster $V(\nu)$ have consecutive *st* numbers (hence y-coordinates), a convex hull of $V(\nu)$ can be computed in $O(|V(\nu)|) = O(n)$ time. Then the total time of computing all convex hulls in $C = (G, T)$ becomes $O(n^2)$. This complexity is slightly reduced as follows. Let $CH(\nu)$ denote the set of vertices which are on the convex hull of a cluster $V(\nu)$ (hence, $|CH(\nu)|$ is the output size of the convex hull). To compute $CH(\nu)$ of a cluster $V(\nu)$, we can discard all vertices that are properly contained inside the convex hull $CH(\mu)$ for some child cluster $V(\mu)$, $\mu \in chl(\nu)$. Before computing the convex hull $CH(\nu)$ for a node $\nu \in \mathcal{V}$, we compute all convex hulls $CH(\mu)$ for the children $\mu \in chl(\nu)$. Then we can compute $CH(\nu)$ from the set $\Delta V(\nu) \cup \bigcup_{\mu \in chl(\nu)} CH(\mu)$ of vertices in $O(|\Delta V(\nu)| + \sum_{\mu \in chl(\nu)} |CH(\mu)|)$ time, where $\Delta V(\nu)$ denotes $V(\nu) - \bigcup_{\mu \in chl(\nu)} V(\mu)$. Therefore, by computing convex hulls from the bottom of the tree T to the root, we can obtain all $CH(\nu)$, $\nu \in \mathcal{V}$ in $O(n + \sum_{\nu \in \mathcal{V}} |CH(\nu)|)$ time, which is linear in terms of the output size of a straight line convex cluster drawing of C.

By Theorem 2, computing a straight-line drawing of the hierarchical graph with n vertices can be done in linear time. A *c-st* numbering of a clustered graph with n vertices can be computed in $O(n)$ time. In summary, we establish the following result on planar straight-line convex cluster drawings.

Theorem 5 *Let $C = (G, T)$ be a c-planar clustered graph with n vertices. A planar straight-line convex cluster drawing of C in which each cluster is a convex hull of points in the cluster can be constructed in $O(n + D)$ time, where $D = O(n^2)$ is the total size of convex polygons for clusters in the drawing.*

Note that as we showed in [14], the results in this section, combining with the techniques in [14], yield an $O(n)$ time drawing algorithm. The algorithm can always provide a planar straight-line drawing for a c-planar clustered graph with each cluster drawn as a trapezoid.

5 Examples and Open Problems

In this section we discuss some drawings produced by our algorithms.

Our algorithm for drawing hierarchical graphs uses a divide and conquer approach. At every division, we choose a vertex v which is drawn as an apex of the polygon; this determines a suitable partition of the polygon. Experiments have shown that the choice of such a vertex v can have a significant impact on the final drawing. For example, consider Figure 21. In Figure 21(a), we always use a vertex v on the right side of the polygon to find a partition. In this case, the final drawing is not quite balanced, although it meets the straight-line and non-crossing requirements. Another drawing of the same hierarchical graph is shown in Figure 21(b). In this case, we have chosen an available vertex v randomly, and the drawing is more balanced.

We have performed the same kind of experiment on clustered graphs. We used rectangular hulls instead of convex hulls to represent a cluster, because we find rectangles are more pleasing, though there is still a chance of edge-region crossings in this case (see Figure 20). We used different partitioning strategies and produced different drawings. From the examples, choosing a vertex v randomly from the available ones seems to be a successful strategy (see Figure 22).

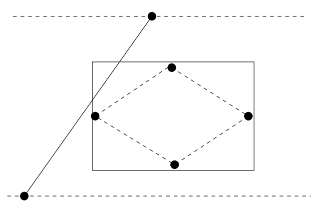


Figure 20: A possible crossing between a segment and a rectangle.

The algorithms presented in sections 3 and 5 use rational numbers for coordinates. In both cases, the precision of the coordinates may increase exponentially with the number of nodes. In other words, if the nodes were placed at integer grid points then the area of the resulting drawing would be exponential. In fact, this is inevitable: exponential area lower bounds have been established both for straight-line h-planar drawings and for straight-line c-planar convex cluster drawings.

- In [37], it is shown that there is a class of hierarchical planar graphs H_n $(n = 1, 2, ...)$ of $4n-1$ layers and $10n - 6$ vertices such that any hierarchical planar straight-line drawing of H_n has width $\Omega((2n-2)!)$ under vertex resolution 1 (that is, every pair of vertices are at least 1 unit distance apart). Figure 23 shows a drawing of H_3.

- In [20], a class of clustered graphs C_n $(n = 1, 2, ...)$ is given. In C_n, there are $2n$ vertices which are partitioned into two clusters. It is shown in [20] that any straight-line convex cluster drawing of C_n has area $\Omega(2^n)$. Figure 24 shows a drawing of C_4.

Future work on hierarchical graphs and clustered graphs should address the following open problems:

- We note that relaxing the straight-line constraints can give us polynomial area bounds both in hierarchical drawings and in convex cluster drawings [1, 19, 13, 14]. In future work, we would like to investigate the trade-off between the number of bends and the area of the drawing.

- The drawings of clustered graphs may lack vertical compaction because we use an *st* numbering as the layer assignment. It is very worthwhile to investigate methods that can improve the vertical compaction.

- In our algorithm, we can only ensure that the non-crossing property holds for clusters drawn as convex polygons. However, it is more desirable to represent clusters as more regular convex bodies such as circles and rectangles. This also forms an interesting topic for our future research.

Acknowledgements

The authors wish to thank Dr. Bryan Beresford-Smith for Figure 1, and the anonymous referees for giving many helpful comments and suggestions.

References

[1] G. Di Battista, W. Didimo, and A. Marcandalli. Planarization of clustered graphs. In *Graph Drawing 2001*, Lecture Notes in Computer Science, pages 60 – 74. Springer-Verlag, 2001.

[2] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: algorithms for the visualization of graphs*. Prentic Hall, 1998.

[3] G. Di Battista, G. Liotta, M. Strani, and F. Vargiu. Diagram server. In *Advanced Visual Interfaces (Proceedings of AVI 92)*, volume 36 of *World Scientific Series in Computer Science*, pages 415–417, 1992.

[4] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61:175–198, 1988.

[5] Claude Berge. *Graphs and Hypergraphs*. North-Holland, 1973.

[6] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. North-Holland, New York, N.Y., 1976.

[7] M.J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-10(11):705–715, 1980.

[8] N. Chiba, K. Onoguchi, and T. Nishizeki. Drawing planar graphs nicely. *Acta Informatica*, 22:187–201, 1985.

[9] N. Chiba, T. Yamanouchi, and T. Nishizeki. Linear algorithms for convex drawings of planar graphs. In J.A. Bondy and U.S.R. Murty, editors, *Progress in Graph Theory*, pages 153–173. Academic Press, New York, N.Y., 1984.

[10] M. Chrobak and T.H. Payne. A linear time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54:241–246, 1995.

[11] P. Eades and X. Lin. How to draw directed graphs. In *Proc. IEEE Workshop on Visual Languages (VL'89)*, pages 13–17, 1989.

[12] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, pages 424–437, 1991.

[13] Peter Eades and Qing-Wen Feng. Drawing clustered graphs on an orthogonal grid. In G. di Battista, editor, *Graph Drawing 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 146 – 157. Springer-Verlag, 1998.

[14] Peter Eades, Qing-Wen Feng, and Hiroshi Nagamochi. Drawing clustered graphs on the orthogonal grid. Journal of Graph Algorithms and Applications, Vol. 3, no. 4, pp. 3-29, 1999.

[15] Peter D. Eades, Xuemin Lin, and Roberto Tamassia. An algorithm for drawing a hierarchical graph. *International Journal of Computational Geometry and Applications*, 6(1):145–156, 1996.

[16] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[17] S. Even and R. E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.

[18] I. Fary. On straight lines representation of planar graphs. *Acta Sci. Math. Szeged.*, 11:229–233, 1948.

[19] Q. Feng. *Algorithms for Drawing Clustered Graphs.* PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, 1997.

[20] Q. Feng, R. Cohen, and P. Eades. How to draw a planar clustered graph. In *COCOON'95*, volume 959 of *Lecture Notes in Computer Science*, pages 21–31. Springer-Verlag, 1995.

[21] Q. Feng, R. Cohen, and P. Eades. Planarity for clustered graphs. In *ESA'95*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer-Verlag, 1995.

[22] Qing-Wen Feng, Peter Eades, and Robert F. Cohen. Clustered graphs and C-planarity. Technical Report 95-04, Department of Computer Science, The University of Newcastle, Australia, 1995.

[23] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[24] E.R. Gansner, S.C. North, and K.P. Vo. Dag – a program that draws directed graphs. *Software – Practice and Experience*, 18(11):1047–1062, 1988.

[25] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weikircher. Advances in c-planarity testing of clustered graphs. In *Proceedings of 10th International Symposium on Graph Drawing*, pages 220–235, 2002.

[26] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.

[27] D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Computing*, 13:338 – 355, 1984.

[28] M. Himsolt. Graphed: An interactive graph editor. In *Proc. STACS 89*, volume 349 of *Lecture Notes in Computer Science*, pages 532–533, Berlin, 1989. Springer-Verlag.

[29] M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. di Battista, editor, *Graph Drawing 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 193 – 204. Springer-Verlag, 1998.

[30] M. Jünger, S. Leipert, and M. Percan. Triangulating clustered graphs. Technical report, 2002.

[31] T. Kamada. *Visualizing Abstract Objects and Relations.* World Scientific Series in Computer Science, 1989.

[32] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996.

[33] M. Kaufmann and D. Wagner. *Drawing Graphs, Methods and Models.* Springer, 2001.

[34] J. Kawakita. The KJ method – a scientific approach to problem solving. Technical report, Kawakita Research Institute, Tokyo, 1975.

[35] W. Lai. *Building Interactive Digram Applications.* PhD thesis, Department of Computer Science, University of Newcastle, 1993.

[36] Thomas Lengauer. Hierarchical planarity testing algorithms. *Journal of ACM*, 36:474–509, 1989.

[37] Xuemin Lin. *Analysis of Algorithms for Drawing Graphs*. PhD thesis, Department of Computer Science, University of Queensland, Australia, 1992.

[38] M. May and P. Mennecke. Layout of schematic drawings. *Syst. Anal. Model. Simul.*, 1(4):307–338, 1984.

[39] Franco P. Preparata and Michael I. Shamos. *Computational geometry: an introduction*. Springer-Verlag, New York, 1985.

[40] H. Purchase. Which aesthetic has the greatest effect on human understanding? In G. di Battista, editor, *Graph Drawing 1997*, volume 1353 of *Lecture Notes in Computer Science*, pages 248 – 261. Springer-Verlag, 1998.

[41] H.C. Purchase, Robert F. Cohen, and M. James. Validating graph drawing aesthetics. In Franz J. Brandenburg, editor, *GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 435–446. Springer-Verlag, 1995.

[42] R. Read. Methods for computer display and manipulation of graphs and the corresponding algorithms. Technical Report 86-12, Faculty of Mathematics, Univ. of Waterloo, July 1986.

[43] L.A. Rowe, M. Davis, E. Messinger, C. Meyer, C. Spirakis, and A. Tuan. A browser for directed graphs. *Software – Practice and Experience*, 17(1):61–76, 1987.

[44] B. Schieber and U. Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM J. Computing*, 17:1253–1262, 1988.

[45] W. Schnyder. Embedding planar graphs on the grid. In *Proc. First ACM-SIAM Symp. on Discrete Algorithms*, pages 138–148, 1990.

[46] Tom Sawyer Software. Graph layout toolkit user's guide. Berkeley, CA, 1996.

[47] S.K. Stein. Convex maps. *Proceedings American Mathematical Society*, 2:464–466, 1951.

[48] K. Sugiyama. A cognitive approach for graph drawing. *Cybernetics and Systems: An International Journal*, 18:447–488, 1987.

[49] K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):876–892, 1991.

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

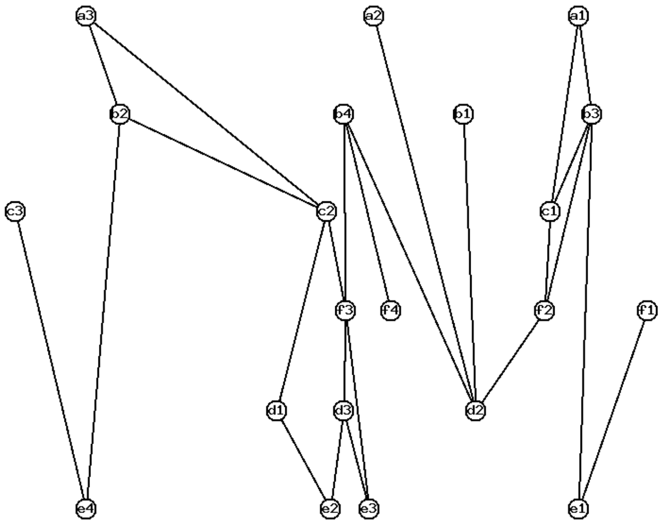
[51] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-18(1):61–79, 1988.

[52] W.T. Tutte. How to draw a graph. *Proceedings London Mathematical Society*, 3(13):743–768, 1963.

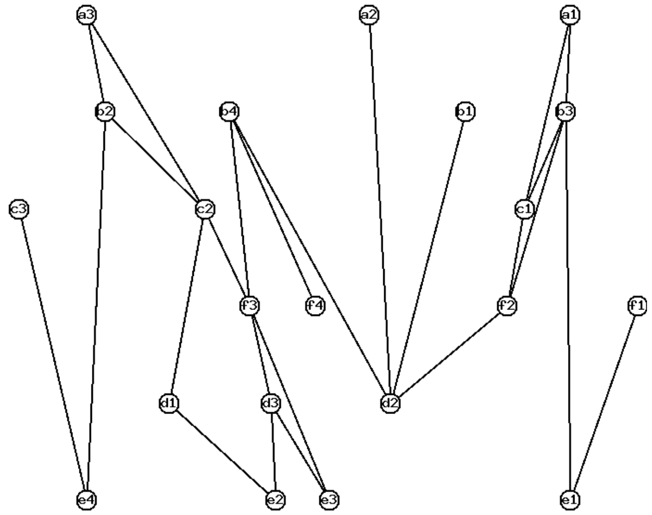
[53] K. Wagner. Bemerkungen zum vierfarbenproblem. *Jber. Deutsch. Math.-Verein*, 46:26–32, 1936.

[54] C. Williams, J. Rasure, and C. Hansen. The state of the art of visual languages for visualization. In *Visualization 92*, pages 202 – 209, 1992.

[55] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object-Oriented Software*. P T R Prentice Hall, Englewood Cliffs, NJ 07632, 1990.

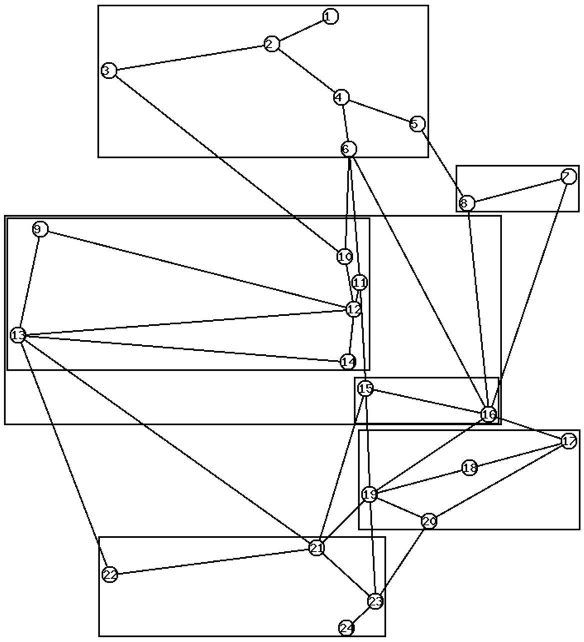


(a)

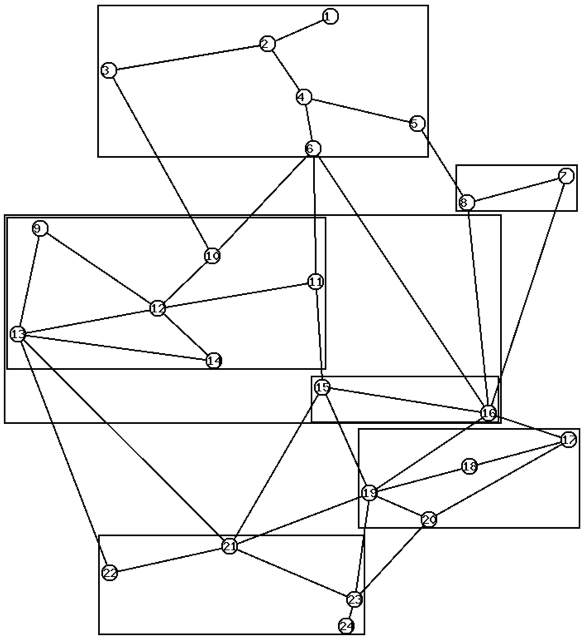


(b)

Figure 21: Example drawings of a hierarchical graph, with different strategies for the partition. (a) Always choose an apex on the right side. (b) Choose an apex randomly from both sides.



(a)



(b)

Figure 22: Example drawings of a clustered graph, with different strategies for the partition. (a) Always choose an apex on the right side. (b) Choose an apex randomly from both sides.

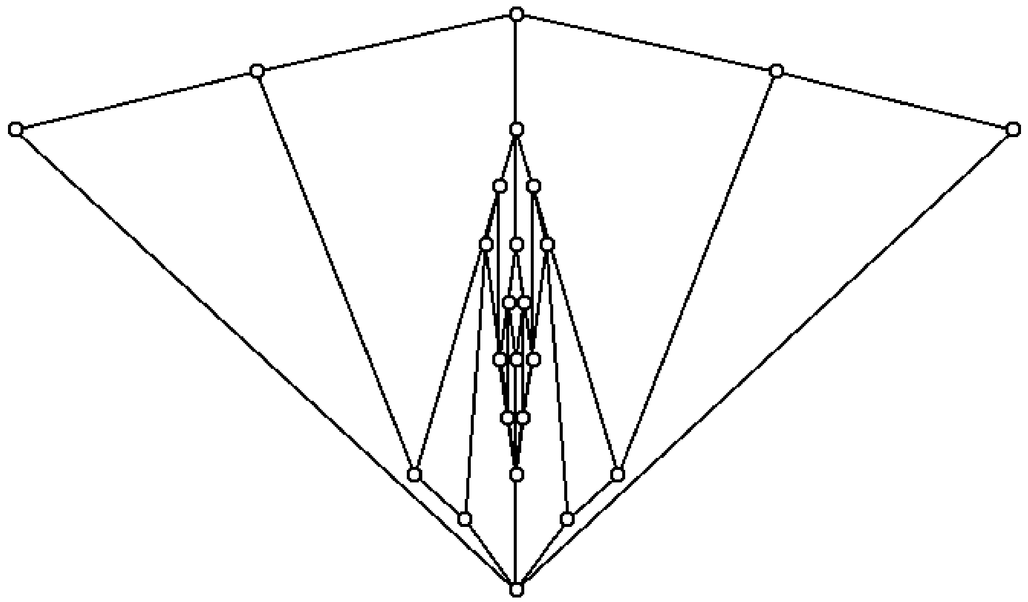


Figure 23: A drawing of H_3.

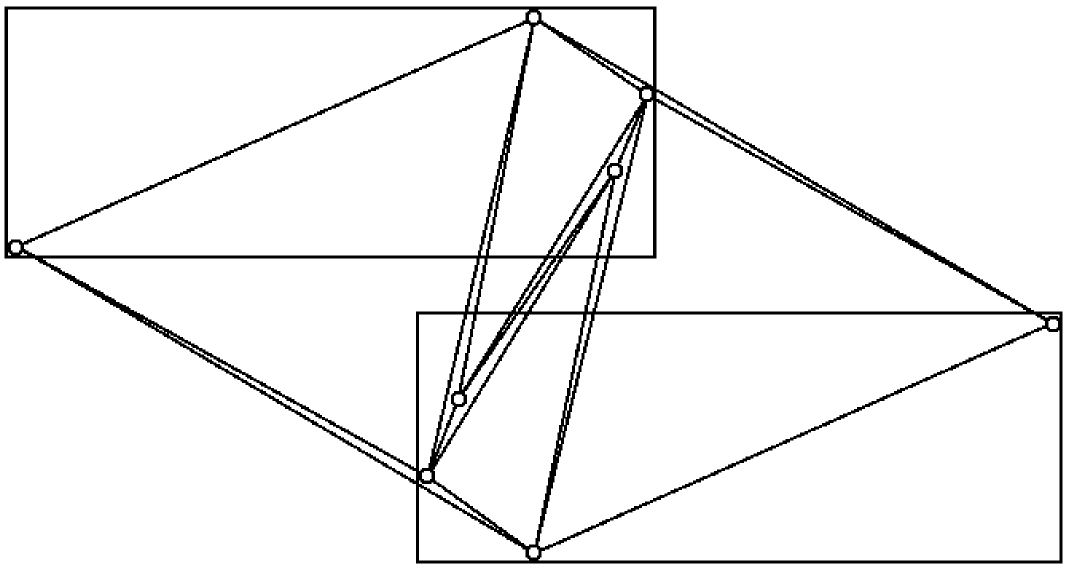


Figure 24: A drawing of C_4.