Journal of Graph Algorithms and Applications
http://www.cs.brown.edu/publications/jgaa/
vol. 3, no. 4, pp. 3–29 (1999)

Drawing Clustered Graphs on an Orthogonal Grid

Peter Eades
Department of Computer Science and Software Engineering
University of Newcastle
http://www.cs.newcastle.edu.au/

Qingwen Feng
Tom Sawyer Software
http://www.tomsawyer.com/

Hiroshi Nagamochi
Department of Applied Mathematics and Physics
Kyoto University
http://www.kyoto-u.ac.jp/

Abstract

Clustered graphs are graphs with recursive clustering structures over the vertices. For graphical representation, the clustering structure is represented by a simple region that contains the drawing of all the vertices which belong to that cluster. In this paper, we present an algorithm which produces planar drawings of clustered graphs in a convention known as *orthogonal grid rectangular cluster drawings*. If the input graph has n vertices, then the algorithm produces in $O(n)$ time a drawing with $O(n^2)$ area and at most 3 bends in each edge. This result is as good as existing results for classical planar graphs. Further, we show that our algorithm is optimal in terms of the number of bends per edge.

Communicated by Giuseppe Di Battista and Petra Mutzel.
Submitted: May 1998. Revised: December 1998 and April 1999.

This work was supported by a research grant from the Australian Research Council and a grant from the Kyoto University Foundation. The work was partially carried out while the second and third authors were visiting the University of Newcastle. Authors' email addresses: eades@cs.newcastle.edu.au, wfeng@tomsawyer.com, naga@kuamp.kyoto-u.ac.jp.

1 Introduction

Graphs are commonly used to model relational information in computing. Many software systems need a graph drawing function. Examples include CASE tools [50], management information systems [22], software visualization tools [49], and VLSI design tools [20]. Graph drawing 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 [7].

With increasing complexity of the information that we want to visualize, more structures are needed on top of the classical graph model. Clustered graphs are graphs with recursive clustering structures (see Figure 1). This type of clustering structure appears in many structured diagrams [20, 26, 28, 38].

Drawing algorithms for clustered graphs are difficult. Heuristic methods for drawing similar structures have been developed by Sugiyama and Misue [30, 39], North [31], and by Madden et al. [25]. Algorithms for constructing straight-line drawings of clustered graphs are given in [9, 10, 15]; note, however, that straight-line drawings of clustered graphs can require exponential area [15].

In this paper, we present a linear time algorithm which produces planar drawings of clustered graphs in a convention called "orthogonal grid rectangular cluster drawings". We apply a technique to order the clusters of the graph recursively, and we use a "visibility representation" for directed graphs to produce our drawings.

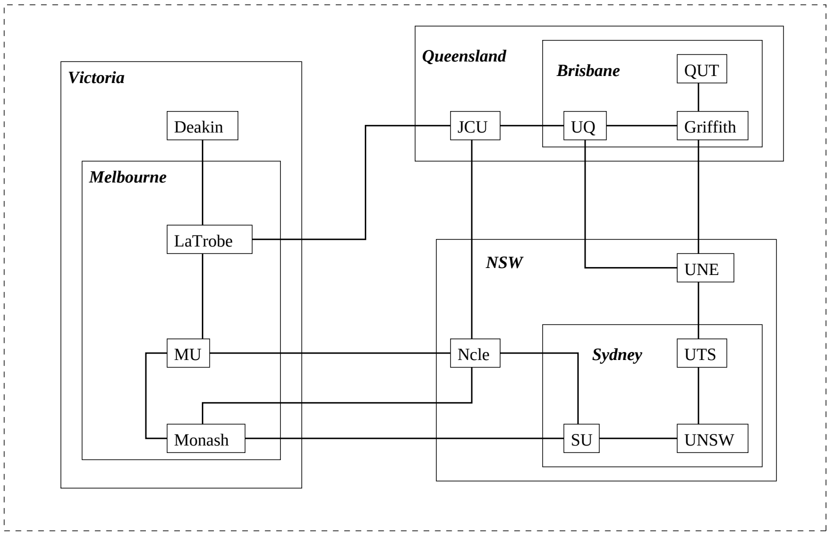

Figure 1: An example of a clustered graph.

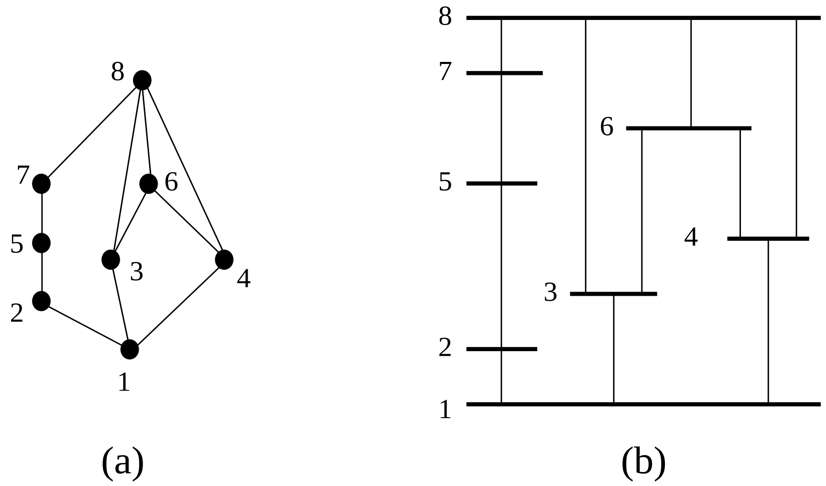

Figure 2: (a) A planar graph. (b) A visibility representation of the graph.

The orthogonal grid drawing convention appears in a number of applications, such as VLSI circuit design [27, 29, 47, 48] and diagrammatic interfaces for relational information systems [1, 42, 32, 35, 40]. Under the orthogonal grid drawing convention, minimizing the number of bends and minimizing the area are the main criteria both for diagram readability and for VLSI design applications.

For classical graphs, several basic results regarding planar orthogonal grid drawings have appeared in the literature. It has been shown by Valiant [48] that any planar graph of degree at most 4 admits a planar orthogonal grid drawing with area $O(n^2)$; further, there are graphs which need quadratic area. Tamassia [41] presented an $O(n^2 \log n)$ time algorithm that computes a planar orthogonal grid drawing with a given planar embedding so that the number of bends is minimized. Garg and Tamassia [19] have shown that if the planar embedding is not given, then the problem is NP-hard.

Several linear time algorithms for planar orthogonal grid drawings of classical graphs have been developed. Tamassia and Tollis [44, 45] have presented an algorithm that outputs drawings with $O(n^2)$ area, where n is the number of the vertices of the graph. If the graph is biconnected, then there are at most $2n+4$ bends in the drawing; otherwise, there are at most $2.4n + 2$ bends. Further, there are at most 4 bends in each edge. If the graph is biconnected, then all but 2 edges have at most 2 bends. Kant [23, 24] has presented an algorithm which improves the result of Tamassia and Tollis in some cases. For triconnected graphs, Kant's algorithm draws on an $n \times n$ grid with at most 2 bends per edge (if $n > 6$), and the total number of bends is no more than $\lceil 3n/2 \rceil + 4$. If the graph is connected with degree at most 3, then the algorithm draws on an $\lfloor n/2 \rfloor \times \lfloor n/2 \rfloor$ grid with at most 2 bends in each edge and no more than $\lfloor n/2 \rfloor + 1$ bends in total. Even and Granot [11] have presented an algorithm such that for any planar graph with degree at most 4, the drawing has $O(n^2)$ area, and there are at most 3 bends in each edge. Lower bounds on the area and the number of bends for planar orthogonal drawings of graphs have been presented by Tamassia, Tollis and Vitter [46], and by Biedl [4].

Another useful representation for planar graphs is the *visibility representation* [36, 43]. Figure 2 shows a planar graph and a visibility representation of the graph. Visibility representation is related to orthogonal drawing in that it is often used as a basis for constructing an orthogonal drawing. Several orthogonal drawing algorithms [11, 44, 45] first construct a visibility representation of the graph, then transform it to an orthogonal drawing.

In Section 3, we present an algorithm for planar drawing of clustered graphs using the same approach. Given an n vertex clustered graph of maximum degree 4, our algorithm produces in $O(n)$ time an orthogonal grid rectangular cluster drawing with $O(n^2)$ area and with at most 3 bends in each edge. This result is as good as the results for classical planar graphs [11, 24, 45]. Further, Section 4 presents a class of graphs each of which has a set of $\Omega(n)$ edges each of which require at least 3 bends; thus there is no algorithm that can improve on the worst case performance of our algorithm with respect to the number of bends per edge. A byproduct of our method is a visibility algorithm for clustered graphs; given an n vertex clustered graph (with no limit on the degree), the algorithm produces a visibility representation where the clusters are represented by rectangles. A clustered graph, together with the visibility representation and orthogonal drawing produced by our algorithm, is in Figure 3. Section 5 concludes with some extensions of our work and some open problems.

2 Terminology

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 node ν in T, let $chl(\nu)$ denote the set of children of ν, and $pa(\nu)$ denote the parent of ν (if ν is not the root). Each node ν of T represents a *cluster* $V(\nu)$ of the vertices of G that are leaves of the subtree rooted at ν. The subgraph of G induced by $V(\nu)$ is denoted by $G(\nu)$. Note that 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 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 simple closed region R that contains the drawing of $G(\nu)$, such that:

(1) the regions for all sub-clusters of ν are completely contained in the interior

of R;

(2) the regions for all other clusters are completely contained in the exterior of R;

(3) 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*

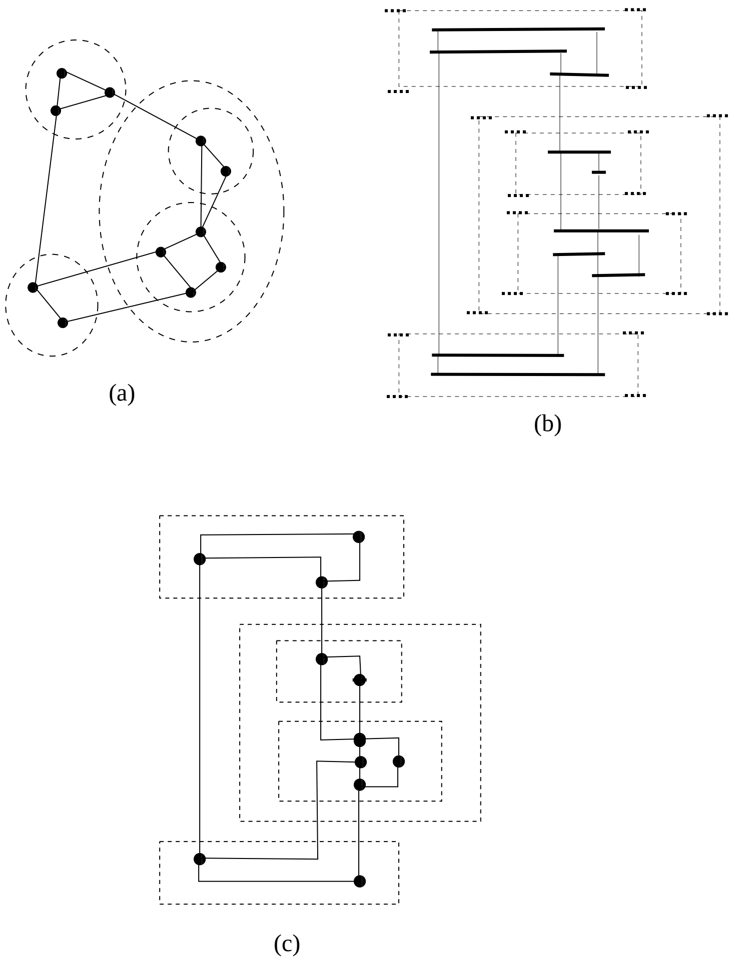

Figure 3: (a) A clustered graph, (b) The visibility representation output by our algorithm, and (c) The orthogonal drawing output by our algorithm.

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*.

A clustered graph $C = (G, T)$ is a *connected clustered graph* if each cluster induces a connected subgraph of G. The following results from [14, 16] characterize c-planarity in a way which can be exploited by our drawing algorithms.

Theorem 1 *A connected clustered graph $C = (G, T)$ is c-planar if and only if the graph G is planar and there exists a planar drawing $\mathcal{D}$ 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 2 *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.*

From Theorem 2, we can assume that we are given a connected clustered graph when drawing a c-planar clustered graph. For each vertex u, a doubly-linked list $A(u)$ of edges around u is given; the edges in $A(u)$ appear around u in the order of the list in the embedding. In the rest of the paper, we further assume that in a clustered graph $C = (G, T)$, every non-leaf node of tree T has at least two children. Hence the size of $T = (\mathcal{V}, \mathcal{A})$ is $O(|\mathcal{V}| + |\mathcal{A}|) = O(n)$.

Our techniques use the concept of planar st-graphs [2]. A planar st-graph is a planar directed graph with one source s and one sink t, such that both the source and the sink above can be embedded on the boundary of the same face, say the external face.

3 Orthogonal Drawings for C-planar Clustered Graphs

An *orthogonal grid rectangular cluster drawing (OGRC drawing)* of a clustered graph maps the graph onto a grid, where edges are drawn as sequences of horizontal and vertical segments, vertices are drawn on grid points, and region boundaries for clusters are drawn as rectangles. Figure 1 is derived from an OGRC drawing. In this section, we present an algorithm that produces a c-planar OGRC drawing for a given c-planar clustered graph $C = (G, T)$ in which each vertex of G has degree at most 4 in G. The drawing has $O(n^2)$ area, and at most 3 bends in each edge. The algorithm works in linear time, and consists of three phases:

(1) visibility representation;

(2) orthogonalization;

(3) bend reduction.

It is clear that OGRC drawings are restricted to clustered graphs in which each vertex of G has degree at most 4 in G; however, the visibility representation (phase 1) can be constructed for any c-planar clustered graph. We treat the three phases in turn.

3.1 Visibility Representation

A *visibility representation* Γ for a planar *st*-graph G maps each vertex v into a horizontal segment $\Gamma(v)$, and each edge e into a vertical segment $\Gamma(e)$ such that:

(1) Segments $\Gamma(u)$ and $\Gamma(v)$ are disjoint for distinct vertices u and v.

(2) If $e = (u, v)$, then the segment $\Gamma(e)$ has its bottom endpoint on $\Gamma(u)$, its top endpoint on $\Gamma(v)$ and does not intersect any other segment.

A *visibility representation* of a clustered graph $C = (G, T)$ consists of a visibility representation of G as well as an isothetic rectangle for each node ν of T, such that the rectangles satisfy the same constraints as the regions for clusters as given in Section 2. An example is in Figure 3(b).

The first phase of our algorithm produces a visibility representation of the input clustered graph. The input clustered graph is enhanced by adding some dummy vertices and edges; this makes a classical graph. A visibility drawing of this classical graph is made; the horizontal lines connecting two horizontal bars (where each horizontal bar represents a dummy vertex) become the horizontal sides of the cluster rectangles, and the vertical lines representing the dummy edges become the vertical sides of the rectangles.

The visibility phase has the following steps:

1. Triangulate an input clustered graph $C = (G, T)$, and compute a specific kind of st-numbering, called a "c-st numbering".

2. Find specific kinds of facial triangles, called "support triangles" for each cluster in the triangulated graph; intuitively, these form the left and right boundaries for each cluster.

3. Extend the original graph $C = (G, T)$ with dummy vertices and edges.

4. Extend the c-st numbering.

5. Find a constrained visibility drawing, and replace the dummy vertices and edges with rectangles.

Our method depends critically on the constrained visibility algorithm of Di Battista, Tamassia and Tollis [3]; this is reviewed and extended slightly in the following subsection. Then we describe each of the steps above in turn.

The algorithm described by the steps above produces a visibility representation of the input clustered graph, and does not require that the input graph has maximum degree 4. However, for Section 3.2, the input must be restricted to maximum degree four, and some alignment constraints must be used to limit the number of edge bends in the final drawing. These alignment constraints are described in subsection 3.1.7.

Some remarks on the construction of the visibility representation are in subsection 3.1.8.

3.1.1 Constrained visibility drawings

The *Constrained_Visibility* algorithm described in [3] takes as inputs a planar st-graph G and a set Π of paths in G, and it produces a visibility drawing of G so that the x-coordinates of $\Gamma(e)$ and $\Gamma(e')$ are the same whenever e and e' are the edges in the same path in Π. The set Π is restricted to "non-crossing paths" in the following sense. Two paths π_1 and π_2 of G are said to be *non-crossing* if they are edge disjoint and do not *cross* at common vertices, that is, there is no vertex v of G with edges e_1, e_2, e_3 and e_4 incident in this clockwise order around v, such that e_1 and e_3 are in π_1 and e_2 and e_4 are in π_2.

For each vertex u in G, the original algorithm *Constrained_Visibility* of [3] chooses the y-coordinate of $\Gamma(u)$ to be the length of the longest directed path from s to u in G. In fact, we can vary the original algorithm to use a topological order λ of an st-graph G as an additional input of *Constrained_Visibility*: the y-coordinate of $\Gamma(u)$ is $\lambda(u)$ for each $u \in V$. Looking at it in a different way, we apply *Constrained_Visibility* to the graph obtained by inserting $\lambda(v) - \lambda(u) - 1$ new vertices in each directed edge (u, v), regarding the resulting paths as part of the set of non-crossing paths. This variation is important for some of the details below.

In our algorithm, the y coordinates of the vertices are the "*c-st* numbers" of the vertices, a specific kind of topological order, as defined in the next subsection.

3.1.2 Computing a c-st numbering

When transforming the clustered graph to a planar st-graph, we need to consider the clustering structure so that the visibility representation that we produce respects the clustering constraints. This is achieved by computing an st-numbering of the vertices of G such that the vertices that belong to the same cluster are numbered consecutively. We call this numbering *c-st numbering*. The *c-st* numbers are used for the y-coordinates of vertices; the consecutiveness ensures that the vertices of each cluster occupy a vertical range. In this paper, a *c-st* numbering $\lambda : V \rightarrow \{1, 2, \ldots, n\}$ of an n vertex clustered graph may have the same number assigned to more than one vertex, as long as it does not violate the property that a vertex $u(\neq s,t)$ is adjacent to two neighbors v and w such that $\lambda(v) < \lambda(u) < \lambda(w)$.

Lemma 1 [9, 10] *Suppose that $C = (G,T)$ is a connected c-planar clustered graph, and G is triangulated. Then a* c-st *numbering of C can be found in $O(n)$ time.*

Proof: The algorithm for obtaining a *c-st* numbering is complex and given fully in [9, 10]; here we briefly sketch the main thrust of the algorithm.

For a cluster $\nu \in \mathcal{V}$, let $G^*(\nu)$ be the (classical) graph obtained from $G(\nu)$ by shrinking each child cluster $V(\nu')$, for each $\nu' \in chl(\nu)$, to a single vertex. An *st* numbering of $G^*(\rho)$, where ρ is the root of the cluster tree T, can be computed in linear time using the algorithm of Even and Tarjan [13]. Now suppose that ν is a child of the root. In this case, we need to augment $G^*(\nu)$ (with vertices and edges from $G^*(\rho)$, $\rho \in chl(\nu)$) and apply the algorithm recursively to the augmented graph. Using the order of the children ν of ρ given by the *st* numbering of $G^*(\rho)$, together with the *c-st* numberings of each $V(\nu)$ provided by recursion, one can obtain a *c-st* numbering of the complete clustered graph. □

With this *c-st* numbering, we transform a clustered graph into a planar *st*-graph by applying directions for edges of G according to the *c-st* numbering.

3.1.3 Find support triangles for each cluster

Next we show how to obtain the 4 bounding sides of the rectangle for a cluster ν. To do this, we need some further terminology. We say that an edge is an *outward edge* (resp., *inward edge*) of a cluster ν if its tail (resp., head) is inside the cluster and its head (resp., tail) is outside the cluster.

As preprocessing, we introduce two new edges (s^*, s) and (t, t^*) with new vertices s^* and t^* outside of G so that each cluster ν has an outward edge and an inward edge. Then we triangulate the graph, as shown in Figure 4, introducing two edges between s^* and t^*; the use of these new edges is described later. The resulting graph G^* is clearly a planar *st*-graph with source s^* and sink t^*.

Note that an outward edge for a cluster ν may well be an outward edge for several clusters, and in fact the sum of the number of outward edges for cluster ν over all clusters ν may be quadratic. We are aiming for a linear time algorithm; to this end we identify a linear number of outward edges with which we can form a rectangle for each cluster. Let $L^+(\nu)$ and $R^+(\nu)$ denote the leftmost outward edge and the rightmost outward edge of a cluster $\nu \in \mathcal{V}$ respectively in G^* (it is possible that $L^+(\nu) = R^+(\nu)$). Similarly, $L^-(\nu)$ and $R^-(\nu)$ denote the leftmost inward edge and the rightmost inward edge of ν respectively.

Identifying the edges $L^-(\nu)$, $L^+(\nu)$, $R^-(\nu)$ and $R^+(\nu)$ helps to define new dummy edges that will form the four bounding sides of the rectangle for a cluster ν. There are at most four of these edges for each cluster, so that the

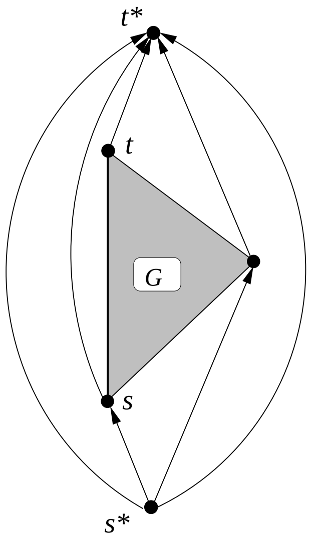

Figure 4: Extending G to G^*.

total number of them is linear. The main difficulty is ensuring that the time required to identify the edges is linear; the remainder of this subsection presents the algorithm for this.

Note that for each node ν in T, $L^-(\nu)$ and $L^+(\nu)$ are contained in the same facial triangle in G^*, since G^* is triangulated. Similarly, $R^-(\nu)$ and $R^+(\nu)$ are in the same facial triangle in G^*. A facial triangle which contains $L^-(\nu)$ and $L^+(\nu)$ (or $R^-(\nu)$ and $R^+(\nu)$) for a cluster ν is a *support triangle* of ν. Any node ν in T has exactly two support triangles (one on its left, and one on its right), where the added edges incident to s^* or t^* ensure the existence of support triangles of the root cluster of G. The support triangles are used to compute the edges $L^-(\nu)$, $L^+(\nu)$, $R^-(\nu)$ and $R^+(\nu)$ for all nodes ν in T. Firstly, however, we must show how to compute the support triangles.

Let $c(u, v)$ denote the least common ancestor of two nodes u and v in T. After $O(n)$ time preprocessing, $c(u, v)$ can be found in $O(1)$ time [21, 37]. Suppose that $\tau = (u_1, u_2, u_3)$ is a facial triangle with directed edges $e_1 = (u_1, u_2)$, $e_2 = (u_2, u_3)$ and $e_3 = (u_1, u_3)$. If ν is a cluster such that $u_2 \in V(\nu)$ and $\{u_1, u_3\} \cap V(\nu) = \emptyset$ then τ is a support triangle of ν; see Figure 5(a). It is easy to see that τ is a support triangle for all clusters ν on the path from u_2 to $\mu = \min\{c(u_1, u_2), c(u_2, u_3)\}$ in T, where the minimum means to take the lower node in T among the two; see Figure 5(b). Thus the following algorithm can be used to identify the support triangles of each cluster:

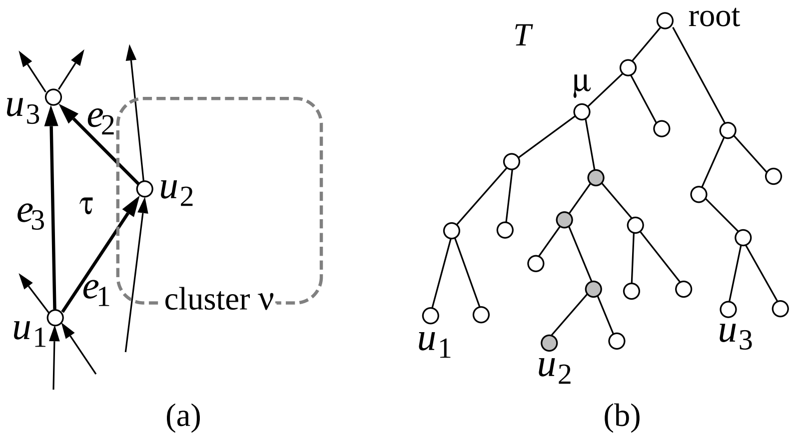

Figure 5: Computing support triangles.

1. Pre-process tree T to prepare for computing $c(u,v)$ for any pair u,v of vertices of G.

2. For each facial triangle $\tau = (u_1,u_2,u_3)$ of G with directed edges $e_1 = (u_1,u_2)$, $e_2 = (u_2,u_3)$ and $e_3 = (u_1,u_3)$:

   (a) compute $\mu = \min\{c(u_1,u_2), c(u_2,u_3)\}$

   (b) Traverse T from u_2 toward the root. Stop the traversal if we reach μ, or if we encounter a node for which a support triangle has been assigned. At each node ν traversed (except for the final node), we assign τ as a support triangle for ν.

From [21, 37], steps 1 and 2(a) take linear time. For step 2(b), note that since each cluster has two support triangles, each node of T is visited at most twice. Thus the total running time is $O(|T|)$, that is, linear.

Given the support triangles, it is easy to compute the edges $L^-(\nu)$, $L^+(\nu)$, $R^-(\nu)$ and $R^+(\nu)$. For each cluster ν, $L^-(\nu) = e_1$ and $L^+(\nu) = e_2$ if $e_2 = (u_2,u_3)$ is clockwise next to $e_1 = (u_1,u_2)$ around u_2; $R^-(\nu) = e_1$ and $R^+(\nu) = e_2$ otherwise. Clearly, the entire running time is $O(|\mathcal{V}|)$, that is, $O(n)$.

3.1.4 Extending the graph with dummy vertices and edges

The next step in the algorithm is to modify $G(\nu)$ in G^* recursively from top to bottom of the tree T, using the edges $L^-(\nu)$, $L^+(\nu)$, $R^-(\nu)$ and $R^+(\nu)$. The head and tail of a directed edge e are denoted by $head(e)$ and $tail(e)$ respectively.

For each cluster ν, we add four dummy vertices $\ell d(\nu)$, $\ell u(\nu)$, $rd(\nu)$ and $ru(\nu)$, then add six new edges (see Figure 6):

$(tail(L^-(\nu)), \ell d(\nu))$,
$(\ell d(\nu), \ell u(\nu))$,
$(\ell u(\nu), head(L^+(\nu)))$,
$(tail(R^-(\nu)), rd(\nu))$,
$(rd(\nu), ru(\nu))$ and
$(ru(\nu), head(R^+(\nu)))$.

The pairs $(\ell d(\nu), \ell u(\nu))$, $(rd(\nu), ru(\nu))$, $(\ell d(\nu), rd(\nu))$ and $(\ell u(\nu), ru(\nu))$ represent the four sides of a rectangle for ν in the final drawing of the clustered graph. In T we place these dummy vertices as children of ν which will be new leaves.

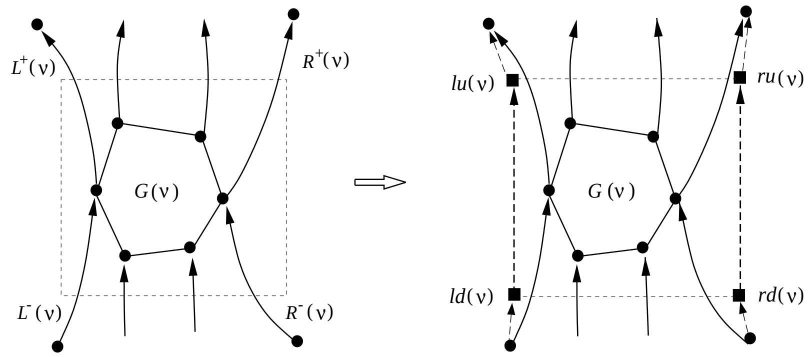

Figure 6: Modify $G(\nu)$, adding dummy vertices to represent the rectangle.

We modify every subgraph $G(\nu)$ recursively as above, obtaining an extended graph F that includes the dummy vertices for the rectangles.

It can be easily verified that the leftmost outward edge of a cluster does not change during the modification for the following reasons.

(i) The edge $L^+(\nu)$ cannot be $R^-(\nu')$ for any other cluster ν' (since the graph G^* is triangulated).

(ii) Since we apply the transformation from top to bottom, adding new edge $(\ell u(\nu), head(L^+(\nu))$ to $G(\nu)$ does not affect the definition of $L^+(\nu')$, $L^-(\nu')$ or $R^-(\nu')$ for any other cluster ν' which is not an ancestor of ν.

Similarly we do not have to recompute the rightmost outward, leftmost inward and rightmost inward edges of a cluster during the modification. It follows that the above modification can be completed in $O(n)$ time.

The resulting graph F is clearly a planar st-graph. It is trivial to add the dummy vertices to T to make an extended cluster tree T^*; this gives an extended clustered graph $C^* = (F, T^*)$. Since we add 4 dummy vertices and 6 dummy edges for every node of T, F has $O(n)$ vertices and edges.

3.1.5 Extending the c-st numbering

We extend the c-st numbering λ in C to a c-st numbering λ' in C^* such that, for each cluster ν:

(i) vertices in $V(\nu) \cup \{\ell u(\nu), \ell d(\nu), ru(\nu), rd(\nu)\}$ are numbered by λ' consecutively, and

(ii) $\lambda'(\ell d(\nu)) = \lambda'(rd(\nu)) < \min\{\lambda'(u) \mid u \in V(\nu)\}$, and $\max\{\lambda'(u) \mid u \in V(\nu)\} < \lambda'(\ell u(\nu)) = \lambda'(ru(\nu))$.

To compute the numbering λ', we consider the tree T of C, where the leaves $u_1, \ldots, u_n$ are arranged from left to right in the order of the c-st numbering λ. We see that, for two consecutive leaves u_i and u_{i+1}, the number δ_i of new dummy vertices w that satisfy $\lambda'(u_i) < \lambda'(w) < \lambda'(u_{i+1})$ is $2\alpha - 1$, where α is the number of internal nodes in the path of T between u_i and u_{i+1}. Thus all δ_i can be obtained by traversing such paths in $O(n)$ time. Based on this, we can easily extend the c-st numbering λ to a c-st numbering λ' of F in $O(n)$ time.

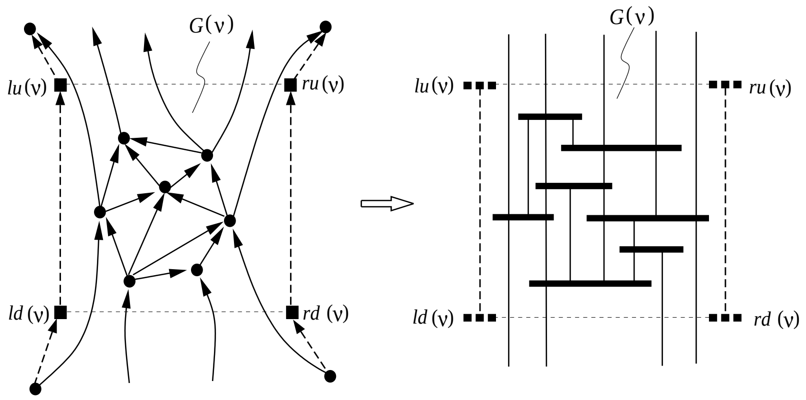

Figure 7: Forming a rectangle for a cluster.

3.1.6 Find a constrained visibility drawing

Using the dummy vertices $\ell d(\nu)$, $\ell u(\nu)$, $rd(\nu)$ and $ru(\nu)$ as corners of rectangles, we can easily deduce the following result.

Theorem 3 *Let $C = (G, T)$ be an n vertex clustered graph with a c-planar embedding. There is a linear time algorithm which constructs a visibility representation of G such that each cluster ν of C can be represented by a rectangle.*

Proof: We obtain a visibility representation Γ of F with the vertices and edges of each cluster ν drawn within a rectangle formed by $\Gamma(\ell d(\nu), \ell u(\nu))$, $\Gamma(rd(\nu), ru(\nu))$ and two line segments $(\Gamma(\ell d(\nu)), \Gamma(rd(\nu)))$ and $(\Gamma(\ell u(\nu)), \Gamma(ru(\nu)))$, as in Figure 7. Since this algorithm preserves the embedding of the input, it is simple to form the rectangles.

Computing the *c-st* numbering takes linear time, and gives the y coordinate of each vertex of G. The algorithm for finding support triangles, given in subsection 3.1.3, also takes linear time. Extending the graph with dummy vertices and edges and extending the c-st numbering is trivial. Finding a visibility drawing (e.g., using the algorithm of [3]) takes linear time. □

Figure 8: Alignment requirement for a vertex v.

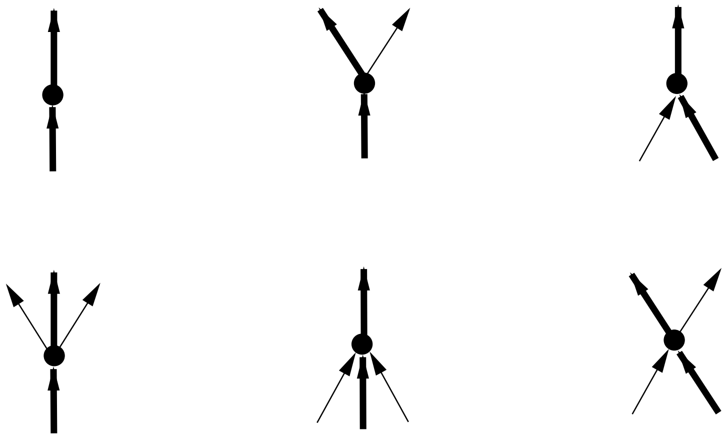

Figure 9: Edge alignment rules.

3.1.7 Edge alignment constraints

Theorem 3 does not use the constraints available in the constrained visibility algorithm. However, looking ahead to the orthogonalization phase, we require that some edges around a vertex are aligned. In the orthogonalization phase, only graphs of maximum degree 4 are considered, and so the edge alignment rules only apply to such graphs. The dummy edges introduced for triangulation are not needed here, and the edge alignment constraints only apply at the original vertices of the graph, and at the vertices $\ell u(v)$, $\ell d(v)$, $ru(v)$, and $rd(v)$, of degree 2.

The purpose of the edge alignment constraints is to avoid unnecessary bends in the orthogonal drawing. For example, suppose that a vertex v has two incoming edges and two outgoing edges; then we require that the right incoming edge is aligned with the left outgoing edge (see Figure 8). The input of the orthogonalization phase is restricted to clustered graphs $C = (G, T)$ for which the vertices in G have maximum degree 4. Thus Figure 9 illustrates all the cases for our alignment requirements; the edges that are marked by thick lines are required to be aligned.

All the above alignment requirements together form a complete specification of the paths that are to be aligned for our visibility representation. Although some of these paths share common vertices, they do not cross with each other. This is because there is at most one path going through each original (non-dummy) vertex of G, and at every dummy vertex, the paths originate from distinct edges of G and therefore do not cross.

We apply the algorithm *Constrained_Visibility* of [3] to the graph F defined in subsection 3.1.4 with a *c-st*-numbering and the above requirements for alignment. In the next phase, we perform orthogonalization operations on this visibility representation.

3.1.8 Remarks on the visibility phase

The representation provided by Theorem 3 does not depend on the maximum degree of the graph; it holds for any c-planar clustered graph. Note also that the triangulation is necessary only for the computation of the *c-st* numbering and the support triangles; the dummy edges inserted to make the triangulation can be omitted after these steps.

3.2 Orthogonalization

For this phase, we assume that the input has maximum degree 4. Note that the process of adding dummy vertices and edges does not increase the maximum degree. To transform the visibility representation to an OGRC drawing, we only need to perform some local operations at each vertex, transforming a horizontal segment to a point. These local operations are illustrated in Figure 10; symmetric cases for (a), (c), (d), (e) and (f) are omitted for brevity. Note that Figure 10 covers all the cases that can appear, since we have required that certain edges around a vertex are aligned. Further, note that a new row is added at every source or sink of degree 4 (see Figure 10(h) and (i)). The number of rows and columns becomes at most twice in in the resulting OGRC drawing.

3.3 Bend Reduction

In the OGRC drawing obtained from the previous phase, an edge is bent only near its endpoints. Hence every edge can have at most 3 bends except in the following case: the edge is between a source of degree 4 and a sink of degree 4 and it has 2 bends near each endpoint (see Figure 11). We show that this 4 bend case can be eliminated by making some adjustments in the drawing.

Note that graph G in a sub-clustered graph $C = (G, T)$ may have multiple sources or sinks. For a source or sink u of degree 4 in G, either the leftmost edge or the rightmost edge gets 2 bends near it, but not both. We say that the

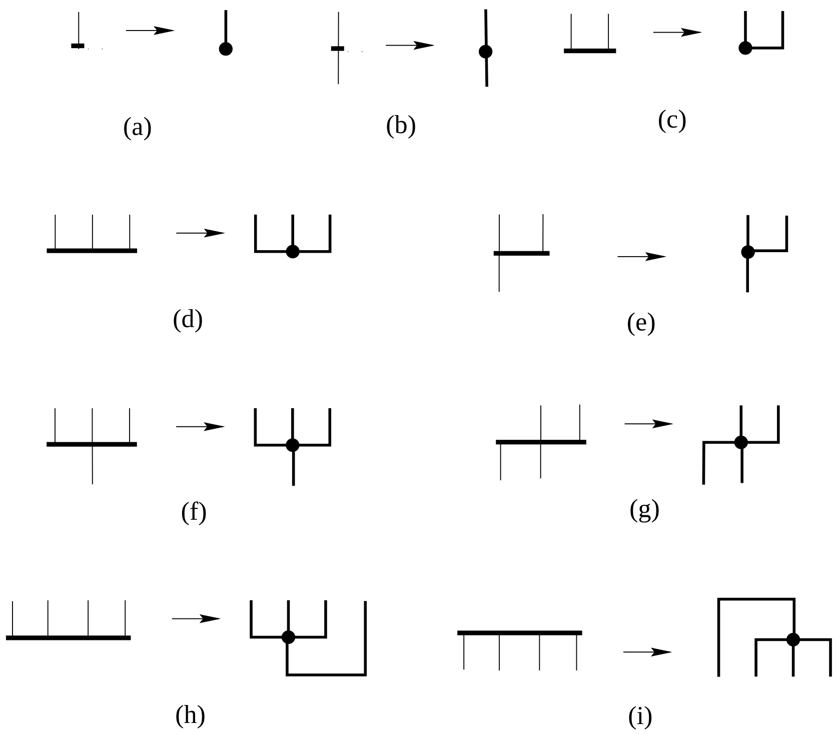

Figure 10: Orthogonalization rules.

leftmost or the rightmost edge of source (or sink) u is an *extreme edge* of u. An edge $e = (u, v)$ is a *critical edge* if e is an extreme edge of both u and v.

Suppose that there is an edge $e = (u, v)$ that has 2 bends near u and another 2 bends near v. Then we fix one end u and rotate the edges around the other end v, letting the edge e bend only once near v (see Figure 12).

However, this operation creates a new bend in the other edge $e' = (v, w)$ of v which may be a critical edge. To avoid this, we construct an auxiliary graph H from G as follows. Let H be the undirected subgraph induced from G by the set of critical edges, where all isolated vertices are deleted and directions of the edges are ignored. Note that the degree of any vertex in H is at most two. This implies that each connected component in H is either a path or cycle. Thus there is an assignment σ of direction of edges such that each vertex has at most one outgoing edge and at most one incoming edge. Let $tail_\sigma(e)$ denote the tail of a critical edge e in terms of such a direction σ. For each critical edge e, we apply the above rotation procedure to $tail_\sigma(e)$ if $tail_\sigma(e)$ has 2 bends near it in the current OGRC drawing. Clearly, after applying the rotation procedure (if necessary) for all tails $tail_\sigma(e)$, there is no critical edge with four bends.

It is easy to see that the entire procedure for reducing bends can be performed in $O(n)$ time and the final OGRC drawing has an $O(n)$ number of rows and columns (and hence the height and width are $O(n)$).

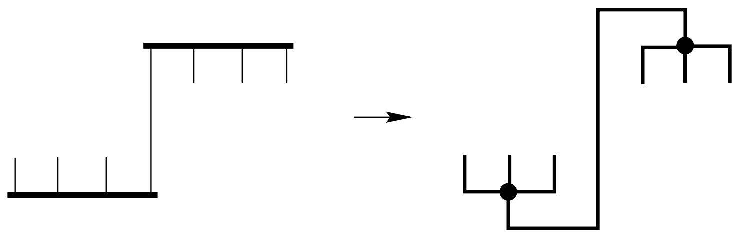

Figure 11: An edge with 4 bends.

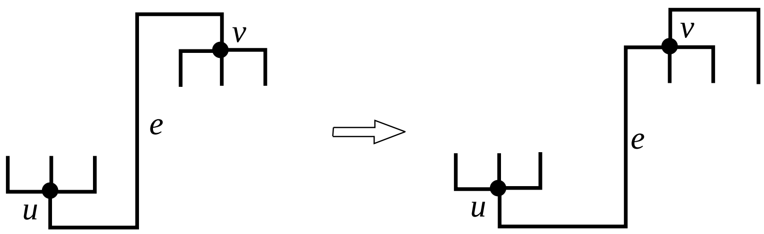

Figure 12: Reduce a bend on a critical edge e.

3.4 The complete algorithm

The algorithm *Orthogonal_Grid_Rectangular_Draw* is described as follows.

Algorithm 1 *Orthogonal_Grid_Rectangular_Draw*
Input: *an n vertex connected clustered graph $C = (G, T)$ of maximum degree at most 4, and a c-planar embedding of C.*
Output: *a c-planar OGRC drawing of C.*

(1) Construct a visibility representation:

  (1.1) triangulate G and compute a *c-st* numbering λ on C;

  (1.2) obtain graph F by modifying G to include the dummy vertices for the rectangles, and compute a c-st numbering λ' of the resulting clustered graph $C^* = (F, T^*)$;

  (1.3) apply algorithm *Constrained_Visibility* to F and λ'.

(2) Construct an OGRC drawing for C from the visibility representation produced from the previous step, using local operations in Figure 10; obtain rectangles for clusters from the visibility representation.

(3) Eliminate all the 4 bend edges by using a rotation procedure.

□

For an n vertex clustered graph $C = (G, T)$, we can triangulate G in $O(n)$ time and compute a *c-st* numbering on C in $O(n)$ time. Since algorithm *Constrained_Visibility* takes linear time [3] in terms of the size of its input, and the graph F has $O(n)$ vertices and edges, step (1) of our algorithm takes $O(n)$ time. Clearly, step (2) takes $O(n)$ time. As noted in Section 3.3, step (3) can be carried out in $O(n)$ time. These make our algorithm take $O(n)$ time. Since graph F has $O(n)$ vertices, edges and bends, the height and width of the output drawing are $O(n)$, as observed in Section 3.3. We summarize the performance of our algorithm in the following theorem.

Theorem 4 *Let $C = (G, T)$ be an n vertex connected clustered graph of maximum degree at most 4, with a c-planar embedding. The algorithm Orthogonal_Grid_Rectangular_Draw constructs in $O(n)$ time a c-planar OGRC drawing of C with $O(n^2)$ area, and with at most 3 bends in each edge.*

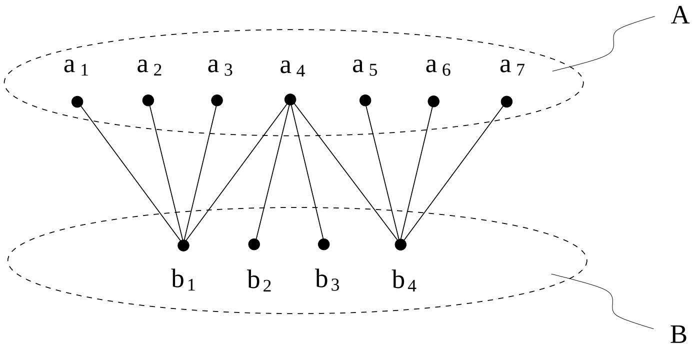

Figure 13: The sub-clustered-graph I.

4 A Lower Bound for Bends

In this section, we present a class of c-planar embedded clustered graphs of n vertices, for which every c-planar OGRC drawing requires $\Omega(n)$ edges bent more than twice. This shows that our algorithm *Orthogonal_Grid_Rectangular_Draw* is optimal in the worst case (in terms of the number of bends in each edge).

To prove our result, let us first consider a small sub-clustered-graph I (see Figure 13), which serves as the building block of our cluster graph. There are two clusters A and B in the sub-clustered-graph I. Cluster A contains vertices $a_1, a_2, \ldots, a_7$; cluster B contains vertices $b_1, b_2, \ldots, b_4$. We assume that I has a fixed c-planar embedding; the orderings of the edges around cluster A and cluster B are shown in Figure 14. The drawings that we discuss in the rest of this section are all consistent with this embedding.

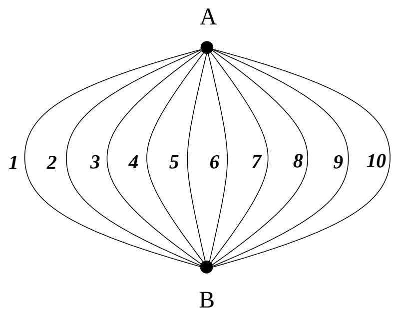

Figure 14: The embedding for clusters A and B.

We prove the following lemma.

Lemma 2 *In every c-planar OGRC drawing of I, there is at least one edge bent more than twice.*

Proof: In a c-planar OGRC drawing of I, cluster A and B are drawn as disjoint rectangular regions. Without loss of generality, we assume that the rectangle for A is drawn above the rectangle for B, and there is a horizontal line ℓ separating them. Note that all the edges between A and B have to cross this horizontal line ℓ, and they cross the line ℓ in the order shown in Figure 14.

Consider the edge (a_4, b_1) and the edge (a_4, b_4), both incident to a_4. Suppose that the edge (a_4, b_1) has no bend above the line ℓ; it follows that the other edge (a_4, b_4) must have more than 2 bends above the line ℓ. On the other hand, if the edge (a_4, b_1) has one bend above the line ℓ, then the other edge (a_4, b_4) must have more than one bend above the line ℓ. We deduce that at least one of these two edges has more than one bend above the line ℓ.

With out loss of generality, let us assume that the edge (a_4, b_4) has more than one bend above the line ℓ. Now consider the edge (a_4, b_4) together with the edge (a_7, b_4). We can show that at least one of these has a total of more than two bends, as follows. If the edge (a_4, b_4) has a bend below the line ℓ, then it has more than two bends in total; if the edge (a_4, b_4) has no bend below the line ℓ, then the other edge (a_7, b_4) must have more than two bends below the line ℓ.

Therefore, we have that there is at least one edge in I that has a total of more than two bends. □

Now we define a class of clustered graphs Φ_n $(n = 1, 2, \ldots)$ with sub-clustered-graph I as the building block. Clustered graph Φ_n consists of a sequence of n copies of the sub-clustered-graph I (see Figure 15). The vertex a_7 of a previous copy of I also serves as the vertex a_1 of the next copy of I. Clustered graph Φ_n has two clusters A_n and B_n. Cluster A_n contains the vertices in the cluster A of each sub-clustered-graph I; cluster B_n contains the vertices in the cluster B of each sub-clustered-graph I. Clearly, Φ_n has $10n + 1$ vertices.

By Lemma 2, we have the following theorem.

Theorem 5 *In every c-planar OGRC drawing of Φ_n $(n = 1, 2, \ldots)$, there are at least n edges bent more than twice.*

5 Remarks

In this paper, we present a linear time algorithm *Orthogonal_Grid_Rectangular_Draw* that produces c-planar OGRC drawings with $O(n^2)$ area and with at most 3 bends in each edge. These results are as good as the results for classical planar graphs [11, 24, 45]. Lower bounds for the area of orthogonal drawings of

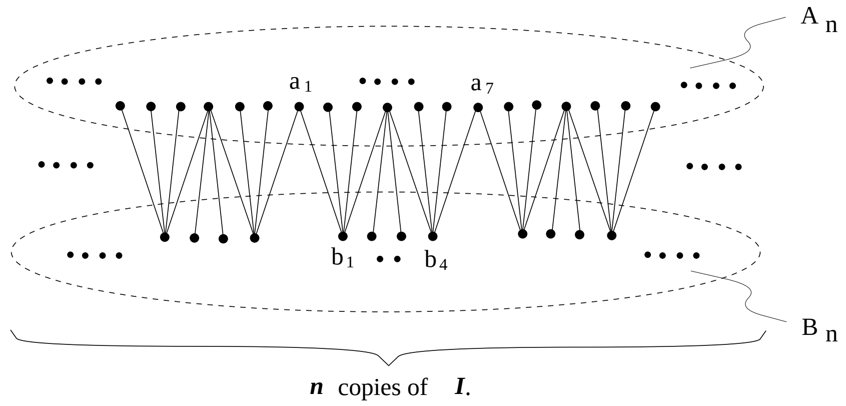

Figure 15: The construction of the clustered graph Φ_n.

classical graphs [48] imply that the area of the drawing produced by our algorithm is asymptotically optimal. Further, we show that the performance of our algorithm is optimal in terms of the number of bends per edge.

Nevertheless, some open problems remain:

- Although the height and the width of our output drawings are both $O(n)$, our algorithm does not guarantee a good aspect ratio. In practice, our algorithm may produce drawings which clearly prefer one dimension against the other; this is because we use a visibility representation which is biased to one dimension. As an example, note that Figure 1 could not be produced by our algorithm as it is. Recent investigations of "2-dimensional visibility representations" [6, 17] of planar graphs (each vertex is represented by a box and each edge is represented by a horizontal or vertical segment between the sides of the boxes) may prove useful in terms of the aspect ratio of the drawing.
- Even and Granot [12] have presented some algorithms for grid layout of block diagrams. Although the drawing requirements there are different from the requirements of drawing clustered graphs, it would be worthwhile to investigate whether we can borrow some of the techniques there and for use in drawing clustered graphs.
- In recent years, many results have been achieved for orthogonal drawings of non planar graphs [5, 4, 33, 34]. It seems very profitable to use these results to extend our algorithm to non planar clustered graphs.
- This paper deals only with graphs of degree at most four. In practice, algorithms must deal with higher degree vertices. Our methods can be

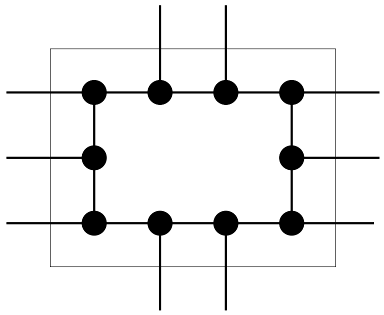

Figure 16: Simulation of a degree 10 vertex with a cluster.

extended to such graphs by simulating the larger degree vertices with clusters; for example, the cluster in Figure16 simulates a vertex of degree 10. It would be interesting to analyze this simulation. However, it is clear that such a simulation increases the area of the drawing. It would be useful to investigate whether the methods for solving the related problem classical graphs [8, 18] can be extended to clustered graphs.

Finally, as a byproduct of our modification from a clustered graph $C = (G, T)$ to an extended clustered graph $C^* = (F, T^*)$ in Section 3.1, we present a new result in planar straight-line drawings of clustered graphs. In a planar straight-line drawing of a clustered graph $C = (G, T)$, edges are required to be drawn as straight-lines and clusters must be drawn as convex polygons. The question of whether every c-planar clustered graph admits a planar straight-line drawing or not has been studied and answered affirmatively [9, 10, 14]. However, it is still open whether or not more regular convex bodies such as circles and rectangles can be used for clusters in a planar straight-line drawing. The results of this paper together with those in [9, 10] imply that a c-planar clustered graph $C = (G, T)$ admits a planar straight-line drawing with clusters drawn as trapezoids, as follows.

We use the extended clustered graph $C^* = (F, T^*)$ of C defined in Section 3.1. Results in [9, 10] imply that for a given c-planar clustered graph $C = (G, T)$ and its *c-st* numbering λ, there is a planar straight-line drawing of C with the following properties:

(i) the y-coordinate of each vertex u is its *c-st* number $\lambda(u)$, and

(ii) the convex polygon for a cluster ν is the convex hull of points in $V(\nu)$.

The drawing can be obtained in $O(n + D)$ time, where D denotes the total size of convex polygons for clusters.

By applying this result to the clustered graph C^* extended from C and a *c-st* numbering λ' on C^*, we can obtain a planar straight-line drawing of C^* in which the convex polygon for each cluster ν is a trapezoid formed by its dummy vertices $\ell d(\nu)$, $\ell u(\nu)$, $rd(\nu)$ and $ru(\nu)$ (since $\lambda'(\ell d(\nu)) = \lambda'(\ell u(\nu))$ and $\lambda'(rd(\nu)) = \lambda'(ru(\nu))$). This implies the following result.

Corollary 1 *Let $C = (G, T)$ be a c-planar clustered graph with n vertices. A planar straight-line drawing of C in which each cluster is drawn as a trapezoid can be constructed in $O(n)$ time.*

Acknowledgments

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

References

[1] C. Batini, L. Furlani, and E. Nardelli. What is a good diagram? a pragmatic approach. In *Proc. 4th Int. Conf. on the Entity Relationship Approach*, 1985.

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

[3] G. Di Battista, R. Tamassia, and I.G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, 41:1–7, 1992.

[4] T. Biedl. New lower bounds for orthogonal graph drawings. In Franz J. Brandenburg, editor, *GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 28–39. Springer-Verlag, 1995.

[5] T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. *Comput. Geom. Theory Appl.*, 9:159–180, 1998.

[6] P. Bose, A. Dean, J. Hutchinson., and T. Shermer. On rectangle visibility graphs. In Stephen C. North, editor, *GD'96*, volume 1190 of *Lecture Notes in Computer Science*, pages 25–44. Springer-Verlag, 1997.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing.* Prentice Hall, Upper Saddle River, NJ, 1999.

[8] Michael Doorley. *Automatic Leveling and Layout of Data Flow Diagrams.* PhD thesis, Department of Computer Science and Information Sys tems, University of Limerick, Ireland, August 1995.

[9] P. Eades, Q-W. Feng, and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. Technical Report 96-02, Department of Computer Science, The University of Newcastle, Australia, 1996.

[10] P. Eades, Q-W. Feng, X. Lin, and H. Nagamochi. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. Technical Report 98-03, Department of Computer Science, The University of Newcastle, Australia, 1998.

[11] S. Even and G. Granot. Rectilinear planar drawings with few bends in each edge. Technical Report 797, Computer Science Department, Technion, Israel Institute of Technology, 1994.

[12] S. Even and G. Granot. Grid layout of block diagrams - bounding the number of bends in each connection. In R. Tamassia and I. G. Tollis, editors, *GD'94*, volume 894 of *Lecture Notes in Computer Science*, pages 64–75. Springer-Verlag, 1995.

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

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

[15] 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.

[16] 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.

[17] U. Fößmeier, G. Kant, and M. Kaufmann. 2-visibility drawings of planar graphs. In Stephen C. North, editor, *GD'96*, volume 1190 of *Lecture Notes in Computer Science*, pages 155–168. Springer-Verlag, 1997.

[18] U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In Franz J. Brandenburg, editor, *GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 254–266. Springer-Verlag, 1995.

[19] A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *GD'94*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 1995.

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

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

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

[23] G. Kant. Drawing planar graphs using the *lmc*-ordering. In *Proc. 33th IEEE Symp. on Foundations of Computer Science*, pages 101–110, 1992.

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

[25] G. Kar, B.P. Madden, and R.S. Gilbert. Heuristic layout algorithms for network management presentation services. *IEEE Network*, pages 29–36, November 1988.

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

[27] M.R. Kramer and J. van Leeuwen. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. In F.P. Preparata, editor, *Advances in Computing Research*, volume 2, pages 129–146. JAI Press, Greenwich, Conn., 1985.

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

[29] C. E. Leiserson. Area-efficient graph layouts (for VLSI). In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, pages 270 – 281, 1980.

[30] K. Misue and K. Sugiyama. An overview of diagram based idea organizer: D-abductor. Technical Report IIAS-RR-93-3E, ISIS, Fujitsu Laboratories, 1993.

[31] S. North. Drawing ranked digraphs with recursive clusters. In *Proc. ALCOM Workshop on Graph Drawing '93*, September 1993.

[32] J. Nummenmaa and J. Tuomi. Constructing layouts for er-diagrams from visibility representations. In *Proc. 9th Int. Conf. on Entity-Relationship Approach*, pages 303–317, 1990.

[33] A. Papakostas and I. G. Tollis. Improved algorithms and bounds for orthogonal drawings. In R. Tamassia and I. G. Tollis, editors, *GD'94*, volume 894 of *Lecture Notes in Computer Science*, pages 40–51. Springer-Verlag, 1994.

[34] A. Papakostas and I. G. Tollis. A pairing technique for area-efficient orthogonal drawings. In Stephen C. North, editor, *GD'96*, volume 1190 of *Lecture Notes in Computer Science*, pages 355–370. Springer-Verlag, 1997.

[35] D. Reiner, G. Brown, M. Friedell, J. Lehman, R. McKee, P. Rheingans, and A. Rosenthal. A database designer's workbench. In S. Spaccapietra, editor, *Entity-Relationship Approach: Proc. 5th Int. Conf. on Entity-Relationship Approach (Dijon France 1987)*, pages 347–360, New York, N.Y., 1987. North-Holland.

[36] P. Rosenstiehl and R.E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete and Computational Geometry*, 1(4):343–353, 1986.

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

[38] 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.

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

[40] R. Tamassia. New layout techniques for entity-relationship diagrams. In *Proc. 4th Int. Conf. on Entity-Relationship Approach*, pages 304–311, 1985.

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

[42] 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.

[43] R. Tamassia and I.G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete and Computational Geometry*, 1(4):321–341, 1986.

[44] R. Tamassia and I.G. Tollis. Efficient embedding of planar graphs in linear time. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 495–498, 1987.

[45] R. Tamassia and I.G. Tollis. Planar grid embedding in linear time. *IEEE Trans. on Circuits and Systems*, CAS-36(9):1230–1234, 1989.

[46] R. Tamassia, I.G. Tollis, and J.S. Vitter. Lower bounds for planar orthogonal drawings of graphs. *Information Processing Letters*, 39:35–40, 1991.

[47] J.D. Ullman. *Computational Aspects of VLSI.* Principles of Computer Science. Computer Science Press, Rockville, Md., 1984.

[48] L. Valiant. Universality considerations in VLSI circuits. *IEEE Transactions on Computers*, C-30(2):135–140, 1981.

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

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