

Multilevel Visualization of Clustered Graphs^{*}

Peter Eades and Qing-Wen Feng

Department of Computer Science and Software Engineering,
University of Newcastle, NSW 2308, Australia
Email: {eades, qwfeng}@cs.newcastle.edu.au

Abstract. Clustered graphs are graphs with recursive clustering structures over the vertices. This type of structure appears in many systems. Examples include CASE tools, management information systems, VLSI design tools, and reverse engineering systems. Existing layout algorithms represent the clustering structure as recursively nested regions in the plane. However, as the structure becomes more and more complex, two dimensional plane representations tend to be insufficient. In this paper, firstly, we describe some two dimensional plane drawing algorithms for clustered graphs; then we show how to extend two dimensional plane drawings to three dimensional multilevel drawings. We consider two conventions: straight-line convex drawings and orthogonal rectangular drawings; and we show some examples.

1 Introduction

Graph drawing algorithms are widely used in graphical user interfaces of software systems. As the amount of information that we want to visualize becomes larger, we need more structure on top of the classical graph model. Graphs with recursive clustering structures over the vertices are called *clustered graphs* (see Fig. 1). This type of structure appears in many systems. Examples include CASE tools [16], management information systems [8], and VLSI design tools [7].

In two dimensional representations, the clustering structure is represented by region inclusions, i.e. a cluster is represented by a simple region that contains the drawing of all the vertices which belong to that cluster (see Fig. 2). For such drawings, some heuristic methods have been developed by Sugiyama and Misue [13, 10], by North [11], and by Madden et al. [12, 9]. Algorithms for planar straight-line convex drawings have been developed by Eades, Feng and Lin [6, 4]. An algorithm for planar orthogonal rectangular drawings is presented by Eades and Feng in [3]. However, as the clustering structure becomes more and more complex, two dimensional representations tend to be insufficient. A common strategy for visualizing large graphs with recursive clusterings is to visualize the graph at multiple abstraction levels. A natural method for such multiple level representations is a three dimensional drawing with each level drawn on a plane at different z-coordinate; and with the clustering structure drawn as a tree in three dimensions. This type of representation not only facilitates visualizing

^{*} This work was supported by a research grant from the Australian Research Council.

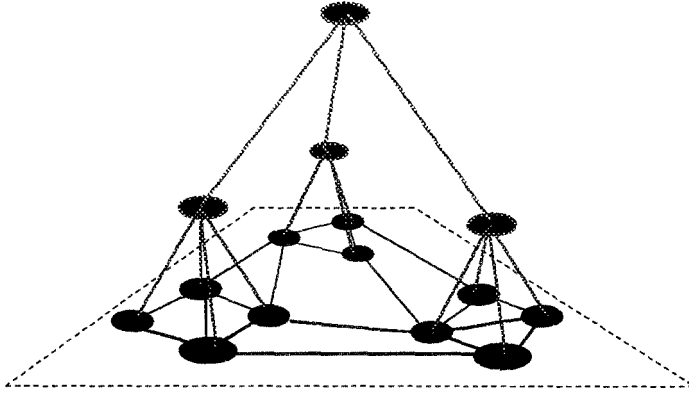


Fig. 1. An Example of a Clustered Graph

the graph at different depth of abstractions, but also keeps the track of the abstractions from one level to another. This is useful in preserving the mental map between abstraction levels.

In this paper, firstly, we describe some two dimensional drawing algorithms for clustered graphs; then we show how to extend two dimensional plane drawings to three dimensional multilevel drawings. We consider two conventions: straight-line convex drawings and orthogonal rectangular drawings; and we show some examples.

2 Terminology

A *clustered graph* $C = (G, T)$ consists of an undirected graph G and a rooted tree T such that the leaves of T are exactly the vertices of G . Each node ν of T represents a *cluster* $V(\nu)$ of the vertices of G that are leaves of the subtree rooted at ν . Note that tree T describes an inclusion relation between clusters. The *height* of a cluster ν , denoted by $h(\nu)$, is defined as the depth of the subtree of T rooted at ν . The *span* of an edge (ν_1, ν_2) of T is $|h(\nu_1) - h(\nu_2)|$. If the span of an edge of T is greater than one, we say it is *long*. In the rest of the paper, we assume every edge of T has a span of one. We consider long edges of T as a sequence of edges, each has a span of one.

For a clustered graph $C = (G, T)$, its *view at level i* is a graph $G_i = (V_i, E_i)$, where V_i consists of the set of nodes of height i in T . There is an edge (μ, ν) in E_i if there is an edge (u, v) of G where u belongs to cluster μ , and v belongs to cluster ν ; in other words, edge (μ, ν) of E_i is the abstraction of all edges between cluster μ and cluster ν in G .

In a *plane drawing* of a clustered graph $C = (G, T)$, graph G is drawn as points and curves in the plane 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:

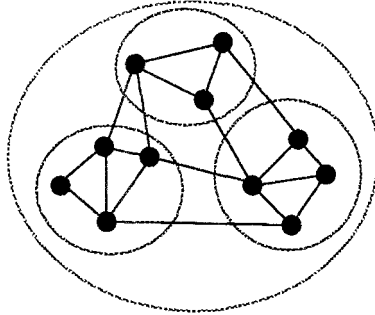


Fig. 2. A 2D Representation of a Clustered Graph

- the regions for all sub-clusters of R 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 plane 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 it is *c-planar* (see Fig. 2). An edge is said to be *incident* to a cluster $V(\nu)$ if one end of the edge is a vertex of that cluster but the other end is not in $V(\nu)$. An *embedding* of a clustered graph consists of the circular ordering of edges around each cluster which are incident to that cluster.

In a *plane drawing of a view* (see Fig. 3), each node is drawn as a simple region in the plane, each edge is drawn as a curve between the region boundaries of its two ends. A plane drawing of a view is *c-planar* if there are no edge crossings or edge-region crossings.

A *multilevel drawing* (see Fig. 4) of a clustered graph $C = (G, T)$ consists of:

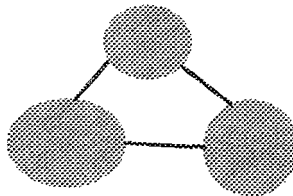


Fig. 3. The Drawing of a View

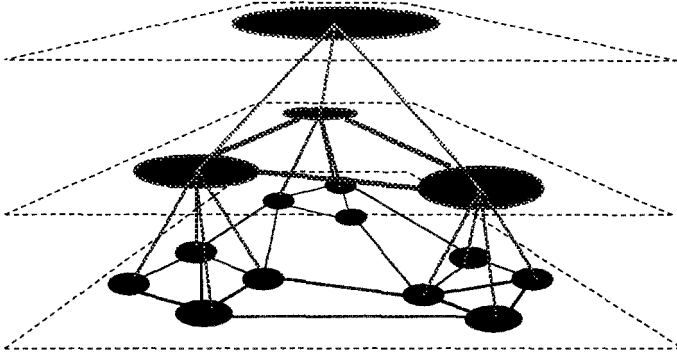


Fig. 4. A Multilevel Drawing

- A sequence of plane drawings of views from the leaf level (level 0) to the root level. The view at level i is drawn on the plane $z = i$.
- A three dimensional drawing of tree T , with each node ν of height i drawn as a point on the plane $z = i$, and within the region of ν in the drawing of the view at that level.

A multilevel drawing of a clustered graph is *c-planar* if the plane drawings of views at all levels are c-planar.

3 Plane Drawings

In this section we describe algorithms which produce c-planar plane drawings of clustered graphs. From these plane drawings, c-planar multilevel drawings can be constructed. We consider two conventions: straight-line convex drawings and orthogonal rectangular drawings.

3.1 Straight-line Convex Drawings

One of the basic graph drawing convention consists of representing edges as straight-line segments. In a straight-line convex drawing of a clustered graph $C = (G, T)$, edges of G are drawn as straight-line segments, regions for clusters are drawn as convex polygons. We use two approaches for such drawings.

An approach based on Tutte's algorithm. This approach from [6] applies a well known algorithm of Tutte [15], which creates a straight-line planar drawing of a triconnected planar graph G such that every face is a convex polygon. To apply Tutte's algorithm, we construct a *skeleton* $\Gamma(\nu)$ for each cluster ν . The skeleton $\Gamma(\nu)$ is the subgraph of $G(\nu)$ consisting of the vertices and edges on the outer faces of the child clusters of ν . Intuitively, a child cluster μ is represented by the outer face of $G(\mu)$ in the skeleton $\Gamma(\nu)$. We recursively apply

Tutte's algorithm to every skeleton graph, and compute a convex polygon for the outer face of each cluster, hence obtain a straight-line convex drawing. However, since Tutte's algorithm works on triconnected planar graphs, this approach is restricted to clustered graphs whose skeletons have the required connectivity property.

An approach based on hierarchical drawings. This approach uses the technique of drawing hierarchical graphs. Hierarchical graphs are directed graphs where vertices are assigned to layers. In a straight-line drawing of a hierarchical graph, vertices assigned to layer i are drawn on the horizontal line $y = i$, arcs are drawn as straight-line segments. If no pair of nonincident arcs intersect in the drawing, we say it is *hierarchical planar (h-planar)*.

In this approach, we transform a clustered graph to a hierarchical graph by computing an *st numbering*¹ of the vertices of G , such that the vertices which belong to the same cluster are numbered consecutively. We call this numbering *c-st numbering*. We use this numbering as a layer assignment to transform a clustered graph to a hierarchical graph, then apply the algorithm presented in [4] to produce a h-planar straight-line drawing.

The *c-st numbering* ensures that each cluster occupies consecutive layers in the drawing. For every cluster, we draw a convex hull of its vertices. It can be shown that in this drawing, there are no edge crossings; and there are no edges that cross the region (the convex hull) of a cluster where they do not belong. Note that if we draw regions as rectangles instead of convex hulls, edge-region crossings are still possible. In fact, by this algorithm, vertices of every cluster are bounded inside a trapezoid region which is formed by two horizontal lines for the highest layer and lowest layer of the cluster, and two straight lines (but not necessarily vertical) on the left and right of the the cluster.

3.2 Orthogonal Rectangular Drawings

In this section, we consider a drawing convention known as *orthogonal rectangular drawings*. In an orthogonal rectangular drawing of a clustered graph $C = (G, T)$, edges of G are drawn as sequences of horizontal and vertical segments, vertices of G are drawn on grid points and regions for clusters are drawn as rectangles. We use a method in [3] which produce such drawings with $O(n^2)$ area, and with constant number of bends on every edge.

Roughly speaking, this method works as follows. First, we transform a clustered graph to a planar *st-graph*², taking into account the clustering structure.

¹ Given any edge (s, t) in a biconnected graph G with n vertices, a *st numbering* for G is defined as follows. 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 and a higher-numbered vertex. Vertices s and t are called the *source* and the *sink* respectively. Such a numbering is an *st numbering* for G . An *st numbering* of a biconnected graph can be computed in linear time [5].

² A planar *st-graph* [1] is a planar directed graph with one source s and one sink t ; and both source and sink above can be embedded on the boundary of the same face, say the external face.

Then we produce a visibility representation of the planar st -graph. Finally, we use orthogonalization method to produce our orthogonal rectangular drawing from the visibility representation.

Here again, we compute a c - st numbering of G . Then we apply a direction for each edge of G according to the c - st numbering, and therefore obtain a planar st -graph. We use the technique in [2] of producing visibility representations of planar st -graphs. To obtain a rectangle for each cluster ν , we add 4 dummy vertices, each represents one side of a rectangle. We also add some dummy edges to obtain the two vertical sides of a rectangle. Then, using the algorithm in [2], we obtain a visibility representation of the graph. Finally, we construct an orthogonal rectangular drawing from the visibility drawing using some local operations similar to [14].

4 Multilevel Drawings

In this section we discuss methods of producing multilevel drawings of clustered graphs. We take the two dimensional plane drawings produced by the algorithms described in the previous section, and we show how to construct three dimensional multilevel drawings from the plane drawings.

To extend plane drawings of clustered graphs to multilevel drawings, we need to consider the following issues:

- Construct the drawing of the view at every level.
- Construct the drawing of the inclusion tree.

To construct the drawing of a view graph, we need to construct the regions for each node of the view, and route every edge between the boundaries of the regions of its two ends. For every node ν of a view at level i , we simply use its representation in the two dimensional plane drawing, and translate them to the plane where $z = i$. Note that every edge (μ, ν) in the view of level i is the abstraction of all the edges that connect between vertices of cluster μ and cluster ν . Therefore, an edge (μ, ν) in the view graph may correspond to multiple edges in G . We choose one edge (u, v) between cluster μ and cluster ν as a representative edge, and derive the drawing of edge (μ, ν) in the view from the drawing of edge (u, v) . Suppose that in the two dimensional plane drawing, cluster μ and ν are drawn as regions $R(\mu)$ and $R(\nu)$ respectively; the drawing of edge (u, v) crosses the boundaries of $R(\mu)$ and $R(\nu)$ at points x and y respectively (see Fig. 5). To construct the drawing of edge (μ, ν) in the view, we use the segment between x and y and translate it to the plane where $z = i$. It can be shown that if the two dimensional plane drawing is c -planar, i.e. with no edge crossings or edge-region crossings, then the drawing derived for each view also has no edge crossings or edge-region crossings. It can also be shown that the derived drawing for each view preserves the convention of the two dimensional plane drawing.

To form the drawing of the inclusion tree T , we need to decide the position of every node, and route the edges between the nodes. Note that a node ν of

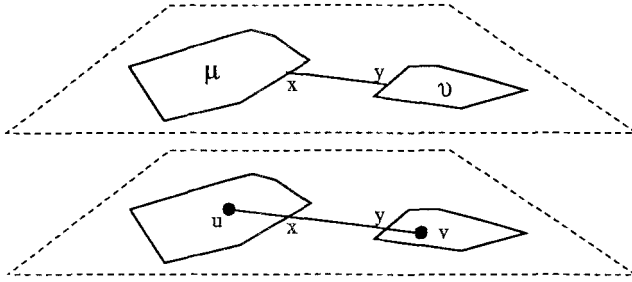


Fig. 5. Forming an Edge in the View

level i has to be positioned on the plane $z = i$ and in the corresponding region of the view. Here, we compute the position of each node recursively from bottom to top of T , as follows:

FOR $i = 0$ to h (the depth of tree T) *DO*

- If $i = 0$, then for each node of level 0 (leaf node), we simply place it at the position where it is drawn in the two dimensional plane drawing.
- For every node ν of level i , we compute the average of the xy -coordinates of its children (at level $i - 1$), and use them as the xy -coordinates for ν .

END

It is easily shown that by this method, every node ν is positioned within the corresponding region in the drawing of the view.

To route the edges of T , we simply draw a straight-line segment between the two nodes. Since we have replaced long edges of T by a sequence of edges, crossings between the edges of T cannot occur. Note that we use the average xy -coordinates of the children as the coordinates of a node. This will put a node right above most of its children and therefore let the edges between a node and most of its children drawn at a large angle to the xy -plane. If a node has only one child, then the edge is strictly vertical. Consequentially, by this method, a long edge of span k is drawn as a line with only one bend. The first segment is strictly vertical and spans $k - 1$ levels. The second segment spans one level.

5 Examples

In this section, we show some examples of drawings produced by our method.

Figure 6 shows a straight-line convex drawing produced using the approach based on Tutte's algorithm. Figure 7 shows the same drawing of Figure 6, but has a viewing direction almost orthogonal to the z axis; this shows the inclusion tree. Figure 8 shows a straight-line convex drawing produced using the approach based on hierarchical drawings. Figure 9 shows the same drawing of Figure 8, but focus on a certain part. This reveals the detail of some substructures. Figure 10

shows an orthogonal rectangular drawing we produced. Figure 11 shows a view inside the orthogonal rectangular drawing of Figure 10; this emphasizes a single level of the abstraction hierarchy.

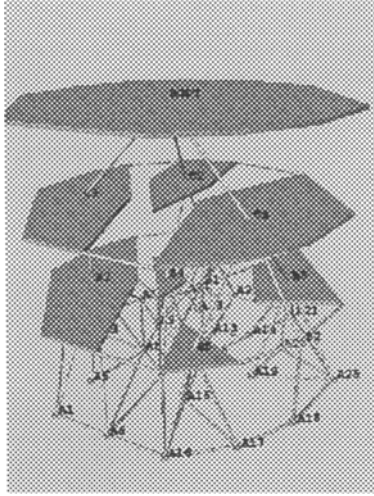


Fig. 6. Example 1

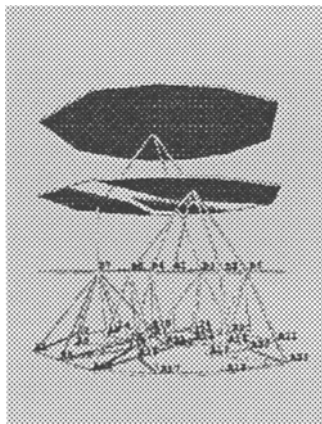


Fig. 7. Example 2

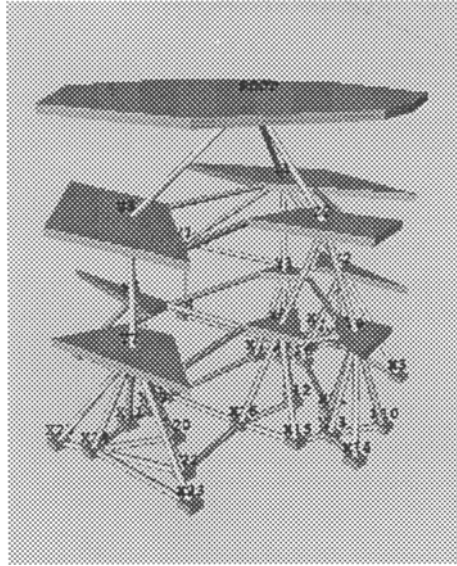


Fig. 8. Example 3

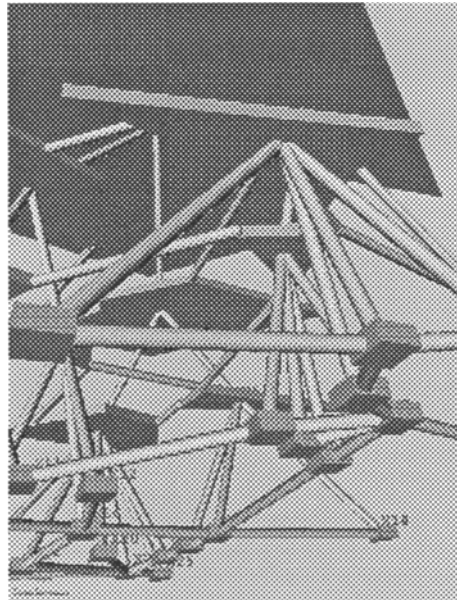


Fig. 9. Example 4

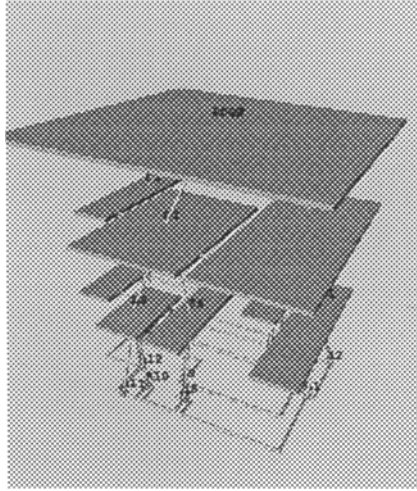


Fig. 10. Example 5

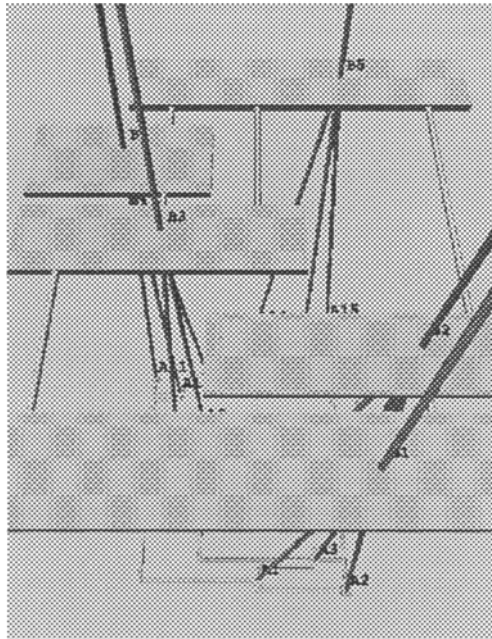


Fig. 11. Example 6

6 Conclusion and Future Work

This paper represents the first attempt to investigate methods for visualizing clustered graphs at multiple abstraction levels and in three dimensions. Particularly, we have considered two drawing conventions: straight-line convex drawings and orthogonal rectangular drawings. We have described some algorithms for two dimensional plane drawings and have shown how to extend them to multilevel three dimensional drawings.

In this paper, every view that we consider is at a specific abstraction level. However, in many applications, we need to visualize a graph at an arbitrary cross-section. For example, sometimes we need to visualize a view with some portions in very detail, and other portions in abstract. This seems an interesting topic for our future research. Further, it will be interesting to investigate the methods of making smooth changes between views based on three dimensional drawings. This would be helpful to some mental map issues in human computer interface design.

Although some of the methods described in this paper may look naive and straightforward, we hope, with the increasing interest in compound structure visualization, more and more results could come forward.

References

1. G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoretical Computer Science*, 61:175–198, 1988.
2. G. Di Battista, R. Tamassia, and I.G. Tollis. Constrained visibility representations of graphs. *Information Processing Letters*, 41:1–7, 1992.
3. Peter Eades and Qing-Wen Feng. Orthogonal grid drawing of clustered graphs. Technical Report 96-04, Department of Computer Science, The University of Newcastle, Australia, 1996.
4. Peter Eades, Qing-Wen Feng, and Xuemin 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.
5. S. Even and R. E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.
6. Qing-Wen Feng, Robert F. Cohen, and Peter 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.
7. D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.
8. J. Kawakita. The KJ method – a scientific approach to problem solving. Technical report, Kawakita Research Institute, Tokyo, 1975.
9. Brendan Madden, Patrick Madden, Steve Powers, and Michael Himsolt. Portable graph layout and editing. In *GD'95*, volume 1027 of *Lecture Notes in Computer Science*, pages 385–395. Springer-Verlag.
10. K. Misue and K. Sugiyama. An overview of diagram based idea organizer: D-abductor. Technical Report IIAS-RR-93-3E, ISIS, Fujitsu Laboratories, 1993.

11. Stephen C. North. Drawing ranked digraphs with recursive clusters. In *Proc. ALCOM Workshop on Graph Drawing '93*, September 1993.
12. Tom Sawyer Software. Graph layout toolkit. available from bmadden@TomSawyer.COM.
13. 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.
14. 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.
15. W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 3(13):743–768, 1963.
16. C. Williams, J. Rasure, and C. Hansen. The state of the art of visual languages for visualization. In *Visualization 92*, pages 202 – 209, 1992.