Layout of Directed Hypergraphs with Orthogonal Hyperedges
(Extended Abstract)*

Georg Sander

ILOG Deutschland GmbH, Ober-Eschbacher Str. 109,
61352 Bad Homburg, Germany
sander@ilog.fr

Abstract. We present a layout algorithm for directed hypergraphs. A hypergraph contains hyperedges that have multiple source and target nodes. Hyperedges are drawn with orthogonal segments. Nodes are organized in layers, so that for the majority of hyperedges the source nodes are placed in a higher layer than the target nodes, similar to traditional hierarchical layout [8,11]. The algorithm was implemented using ILOG JViews[10] for a project that targeted electrical signal visualization.

1 Introduction

While classical graphs deal with edges between pairs of nodes (or vertices), a hypergraph deals with hyperedges between more than two nodes. In mathematical terms, a directed hypergraph $G = (N, E)$ contains a set N of nodes and a set $E \subseteq \mathcal{P}(N) \times \mathcal{P}(N)$ of hyperedges. A hyperedge $e = (S, T)$ has source nodes $S \subseteq N$ and target nodes $T \subseteq N$.

In the literature, various drawing strategies are described for *undirected* hypergraphs [5,6,7]. To lay out *directed* hyperedges, some graph drawing tools use traditional polyline graph layout techniques applied on regular edges, and simulate the hyperedge by overlapping the start line segments of all regular edges that represent the hyperedge [9,10]. A disadvantage of this approach is that the paths of the simulated hyperedge branches out very early, so that the hyperedge drawings are unnecessary complex. We show in this article a better solution.

In a project for the automobile industry, we developed a display tool for schematics of automotive communication networks. To show different views to such a communication network and to show different levels of details we provided three different layout algorithms. This paper sketches the foundations of one of these algorithms which draws so-called *function net diagrams* of a communication network. The nodes in a function net diagram represent control units for, e.g., outside mirrors, turn signals, automatic break assistants, or horns. Connections between the nodes represent electric signals that are emitted from a single

* The full article is available via ftp://ftp.ilog.fr/private/ILOG.de/rnd/gsander/public/hypergraph.ps.gz

G. Liotta (Ed.): GD 2003, LNCS 2912, pp. 381–386, 2004.

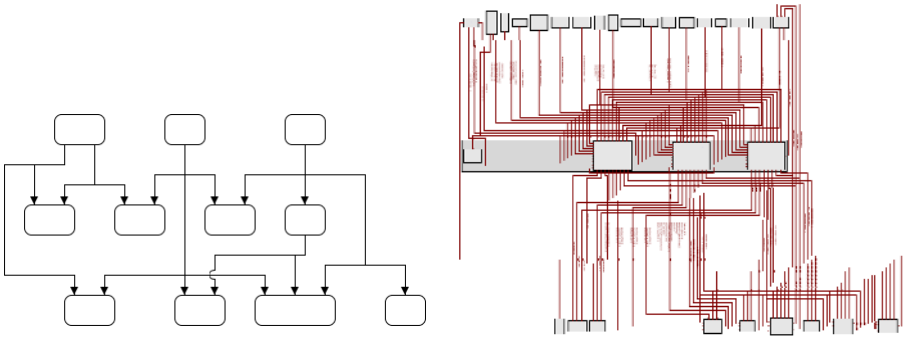

Fig. 1. Hypergraph with orth. hyperedges (left: schematic, right: real world diagram)

function and are received by one or several other functions. For example, a signal containing distance information is submitted to a park distance control unit.

An electrical signal is essentially a single source hyperedge: it starts at the electrical component that produces the signal, and it ends at all components that need the signal as input. The drawing conventions for these diagrams include that the hyperedge is drawn by a set of orthogonal line segments so that there is a path along the segments from the source node to each target node. Each path towards one target node should share many segments with the paths to the other target nodes. (Fig. 1). The customer requirements include that the components (nodes) are organized in horizontal layers. Additional goals are to balance the diagram and to avoid unnecessary crossings, segment overlaps and bends.

The traditional hierarchical layout [8,11] has similar layout objectives; however, it is designed for classical graphs and does not work for hypergraphs. In the following, we assume that the reader is already familiar with the details of traditional hierarchical layout. We sketch the new additions that are necessary to convert the traditional hierarchical layout into an algorithm suitable for hypergraphs. Even though the project used single source hyperedges, our algorithm can be applied to general directed hyperedges as well.

2 Grid Representation

The first goal is to position the nodes on a preliminary two-dimensional grid. Each node consumes one grid cell, i.e., it obtains an integer grid coordinate (i, j) where i is the column number and j is the row number of the grid. The grid representation easily allows the routing of long hyperedge segments that span several columns or several rows without bends because the nodes are aligned to the grid cells so that there is free space between rows and columns (Fig. 2). In order to produce a more balanced drawing, the layout will dissolve the preliminary grid in a later step.

Traditional hierarchical layout deals with layers, which corresponds to rows in this grid. For each hyperedge, all source nodes should be in a higher layer

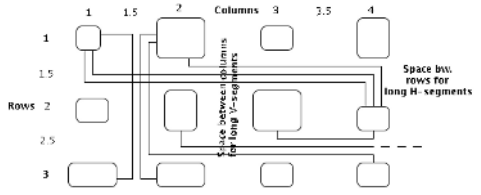

Fig. 2. Grid representation of the hypergraph

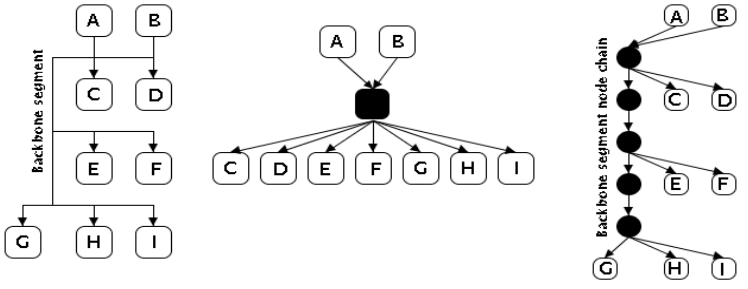

Fig. 3. Left: hypergraph with 1 hyperedge, mid: layering graph, right: cross.red.graph

than all target nodes (source-target condition). Therefore, we translate the hypergraph into a layering graph. Besides hypergraph nodes, each hyperedge e has a representative node n_e in the layering graph. For each hyperedge e, regular edges are added to the layering graph from the source nodes to n_e and from n_e to the target nodes (Fig. 3 middle). The layering graph is an appropriate representation of the source-target relationship of the hyperedges. If the layering graph is cyclic, then some hyperedges must violate the source-target condition. Since the layering graph is a regular graph, we use the standard heuristic to make the graph acyclic and to calculate a ranking of the nodes [1]. From the ranking, the layer number (row number) can be obtained by eliminating all ranks that contain only hyperedge nodes n_e but no regular nodes.

In order to calculate the column number, we need an appropriate relative ordering of the nodes within the layers. This can be considered as preliminary crossing reduction, because a good ordering ensures that later a routing with only few crossings is possible. Each hyperedge that spans several layers must have a backbone segment from which the segments can branch out that connect the end nodes. We translate the hypergraph into a crossing reduction graph (Fig. 3 right). The crossing reduction graph differs from the layering graph in the representation of the hyperedges:

- We duplicate the number of layers: each node at layer j corresponds to a node at layer $2j$ in the crossing reduction graph. The odd layer numbers are reserved for nodes representing hyperedges.
- The backbone segment of a hyperedge is represented by a chain of nodes in the crossing reduction graph that spans from $j_{\min} + 1$ to $j_{\max} - 1$, where

$j_{\min}$ and $j_{\max}$ is the minimal and maximal layer number of an end node of the hyperedge in the crossing reduction graph. All nodes of the chain are sequentially connected by regular edges in the crossing reduction graph.
- Each source node n of the hyperedge at layer j is connected by a regular edge to the node of the backbone segment chain at layer $j+1$.
- Each target node of the hyperedge at layer j is connected to the node of the backbone segment chain at layer $j-1$.

The crossing reduction graph is a proper hierarchy, hence the standard techniques [8] for crossing reduction in hierarchical layout can be applied to obtain the relative ordering of nodes within levels.

3 Creating the Hyperedge Segments

Once all nodes have integer row and column coordinates, we create the hyperedge segments so that they run between the node grid coordinates. All horizontal segments that must be placed in the free space between row j and $j+1$ get the preliminary coordinate $j+0.5$, and similarly with the vertical segments.

For each hyperedge, first the vertical backbone segment is created, and then the horizontal segments that run between the rows are connected as needed to the backbone segment. For each hyperedge, only maximally one horizontal row segment per row is needed. Finally, the row segments are connected to the source and target nodes of the hyperedge via vertical end segments. The preliminary coordinate assignment for all segments is straight forward, except for the backbone segment. We use a heuristic for the preliminary coordinate of the backbone segment. The heuristic has the goal to reduce the number of segment crossings. The full article illustrates the details. As result, all hyperedge segments are created, but segments of different hyperedges may overlap, since only segment coordinates 0.5, 1.5, 2.5, etc, are used.

4 Disentangling Segment Coordinates

To resolve the segment overlaps, all segments that run between the same grid row or grid column are collected. For instance, all horizontal segments at coordinate $k+0.5$ are collected in a set S_k^H. The goal is to spread them to different coordinates between k and $k+1$. The distribution of the segments influences the number of segment crossings (Fig. 4). A simple yet efficient solution of the problem based on sifting is illustrated in [3]. We sketch here quickly a different, more sophisticated solution: To minimize the number of crossings, we generate the segment crossing graph for S_k^H. Each segment of S_k^H corresponds to a node in the segment crossing graph. For each pair of segments $s_1, s_2 \in S_k^H$, we calculate the number of crossings C_1 if s_1 has a lower coordinate than s_2, and C_2 if s_2 has a lower coordinate than s_1, If $C_1 < C_2$, we add an edge between s_1 and s_2 with cost $C_2 - C_1$, otherwise we add an edge between s_2 and s_1 with cost $C_1 - C_2$.

If the segment crossing graph is acyclic, the segment crossing graph can be sorted topologically. If the final segment coordinates respect this topological

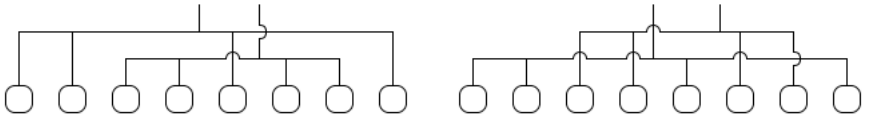

Fig. 4. Two distributions of the same segments. Left: 2 crossing, right: 5 crossings

ordering, then the number of crossings is minimal and no segments overlap. However, usually, the segment crossing graph contains cycles. In this case, some edges must be removed from the segment crossing graph to break the cycles, using one of the standard techniques for this problem [1].

Finally, we calculate a ranking of the segments from the acyclic segment crossing graph. Segments of S_k^H with rank $r \in \{1, \ldots, r_{\max}\}$ get the coordinate $k + r/(r_{\max} + 1)$, hence are spread between k and $k + 1$. After this step is done for all sets S_k^H of horizontal segments, the same step is done for all sets S_k^V of vertical segments.

5 Balancing

Nodes aligned to a grid waste space. Furthermore, the layout is not yet well balanced, because the column numbers of the nodes were calculated from the ordering of the nodes in the crossing reduction graph, and the crossing reduction graph does not take any balance criteria into account. The requirements of the application included that the nodes are organized in layers but not on a grid. Hence we dissolve the grid now. We keep the rows (layers) but remove the columns, by allowing the nodes to be shifted within the layers to any non-integer coordinate. The balancing rules require for instance that a source node is centered above the target nodes (Fig. 5).

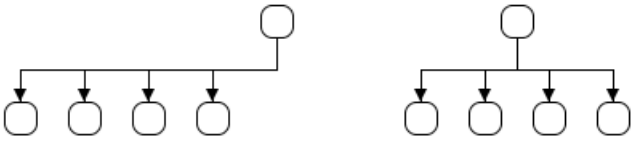

Fig. 5. Left: unbalanced situation. Right: source node is balanced.

In [8], we introduced the pendulum method to balance a traditional hierarchical layout. This method is extended for hypergraphs. The central question of the method is how far a node or segment can be shifted without overlapping the neighbored objects. In a classical, layered graph, this question is trivial to answer. In a hypergraph, it is more difficult, because long vertical segments may be influenced by nodes on different levels. Therefore, a visibility graph [2] is constructed. Edges in the visibility graph indicate the neighbor relationship

between nodes and segments. The modification of the pendulum method with the visibility graph is described in the full article.

6 Conclusion and Acknowledgment

A variant of the layout algorithm was implemented by using the ILOG JViews Component Suite in Java [10]. As with the traditional hierarchical layout, many subproblems of the hypergraph layout are NP complete [4], and heuristics are used. The layout speed is sufficient, however it is slightly slower than traditional hierarchical layout, due to the fact that the treatment of hyperedges is more complex than the treatment of regular edges.

The layout algorithm was developed in a project funded by BMW. We thank the project team at ILOG: F. Baumann, X. Loiseau, C. Sirvain, F. Stork, and J.P. Vandara. Furthermore the close collaboration with Mr. Schumm and Dr. Schuller (both BMW) is appreciated.

References

1. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice-Hall, Inc., New Jersey, 1999.
2. G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic digraphs. *Theoret. Comput. Sci.*, 61:175–198, 1988.
3. T. Eschbach, W. Günther, and B. Becker. Crossing reduction for orthogonal circuit visualization. In *Proc. International Conference on VLSI, Las Vegas*, pages 107–113. CSREA Press, 2003.
4. M. R. Garey and D. S. Johnson. Computers and intractability: A guide through the theory of NP-Completeness. *W. H. Freeman, New York, NY*, 1979.
5. H. Gropp. The drawing of configurations. In *Proc. Symposium on Graph Drawing, GD'95*, pages 267–276. Springer, LNCS 1027, 1996.
6. D. S. Johnson and H. Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987.
7. E. Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34:177–185, 1990.
8. G. Sander. Graph layout through the VCG tool. In *Proc. DIMACS International Workshop on Graph Drawing, GD'94*, pages 194–205. Springer, LNCS 894, 1995.
9. G. Sander. A fast heuristic for hierarchical Manhattan layout. In *Proc. Symposium on Graph Drawing, GD'95*, pages 447–458. Springer, LNCS 1027, 1996.
10. G. Sander and A. Vasiliu. The ILOG JViews graph layout module. In *Proc. Symposium on Graph Drawing, GD 2001*, pages 438–439. Springer, LNCS 2265, 2002.
11. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Sys. Man, and Cybernetics*, SMC 11(2):109–125, 1981.