Drawing Hypergraphs in the Subset Standard
(Short Demo Paper)

François Bertault and Peter Eades

[1] Tom Sawyer Software, 804 Hearst Avenue, Berkeley, CA 94710, USA.
bertault@tomsawyer.com
[2] The University of Sydney, F09 - Madsen, NSW 2006, Australia.
eades@cs.usyd.edu.au

Abstract. We report an experience on a practical system for drawing hypergraphs in the subset standard. The PATATE system is based on the application of a classical force directed method to a dynamic graph, which is deduced, at a given iteration time, from the hypergraph structure and particular vertex locations. Different strategies to define the dynamic underlying graph are presented. We illustrate in particular the method when the graph is obtained by computing an Euclidean Steiner tree.

1 Introduction

Hypergraphs can be viewed as an extension of classical graphs in which edges can represent relationships between more than two vertices. Figure 1 is an example of a possible graphical representation of an hypergraph, where inclusion relationships are represented by curves that enclose node. An interpretation of the drawing could be that Paul and John are familiar with GUIs, Paul and Chris with OSs, Alex, Mary and Bob with SEs, while only Mary and Chris know about DBase. Sadly, only a few papers in the literature studies the drawing of hypergraphs [4,7,6].

Johnson and Pollak introduced two notions of planarity of hypergraphs, inspired by the Venn diagram representations of sets [9], and have given NP-completeness results in [4]. In the *hyperedge-based Venn diagrams* and *vertex-based Venn diagrams*, hyperedges are represented by faces of a planar graph that satisfy some connectivity property.

Mäkinen introduced two kinds of hypergraph drawings in [7]. In both cases, vertices are represented by points in the plane. In the *edge standard*, an hyperedge e is represented by connecting the points that represent the vertices that define e by smooth curve lines. Two vertices belong to the same hyperedge if there is a smooth curve between the points that represent these vertices. In the *subset standard*, an hyperedge is represented by a closed curve that contains exclusively the points that represent the vertices that define the hyperedge. Figure (2,left) is an example of drawing of an hypergraph using these two representations. A method for drawing hypergraphs in the edge standard is given in [6].

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 164–169, 2001.

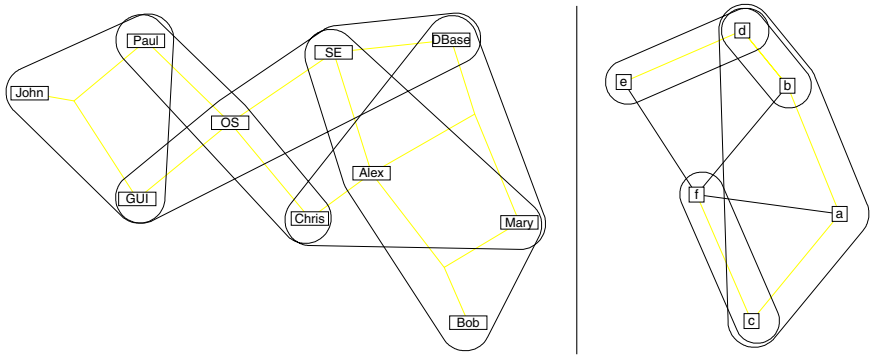

Fig. 1. A simple hypergraph (left) and a higraph (right), drawn with PATATE .

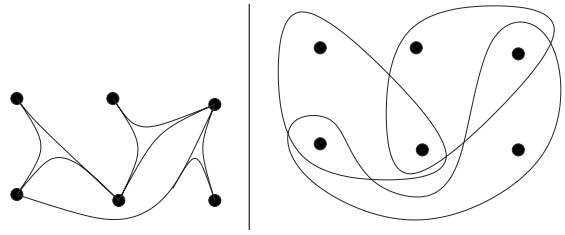

Fig. 2. Drawing of a hypergraph in the edge standard (left) and in the subset standard (right).

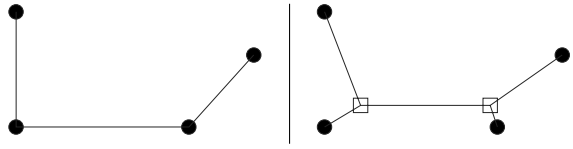

Fig. 3. Example of minimum Euclidean spanning tree (left) and Euclidean Steiner tree (right).

Our PATATE system focuses on the representation of hypergraphs in the subset standard. In this representation, the emphasis is on the representation of hypergraphs as set intersections. This representation is also a step toward the drawing of higraphs, structures introduced by Harel in [3]. The higraph model can be viewed as an hypergraph structure in which edges between nodes or hyperedges can be added. PATATE can already handle drawings for a subset of higraphs; the edges can be defined between nodes only, and are then represented by straight lines. Figure (2,right) is an example of such a higraph drawing obtained with PATATE.

2 The Method

A *graph* $G = (V, E)$ is defined by a finite set V of *vertices* and a finite set E of *edges*, that is, unordered pairs of vertices. A *hypergraph* $H = (V, E)$ is defined by a finite set V of *vertices* and a finite set E of *hyperedges*, that is, unordered non-empty finite sets of vertices. In this paper, we consider *simple* hypergraphs, that is hypergraphs that contain hyperedges between at least two elements.

The method implemented in PATATE is as follows:

1) Assign random locations to vertices of H
2) For a given number of iterations
   a) Construct a graph G from the current positions of the nodes of hypergraph H, using one of the three methods described below. For every vertex v in H, there will be a vertex $\nu(v)$ in G.
   b) Set the locations of the vertices in G to be the locations of the associated vertices in H according to ν.
   c) Apply a force directed drawing algorithm to compute new locations for the nodes in G.
   d) Set the locations of the vertices in H to be the locations of the associated vertices in G.
3) For each hyperedge $e = \{v_1, \ldots, v_k\}$ in H, build a curve that is obtained as the contour of the union of the edges in G that have both ends in e; the union of two edges is defined by the union of the drawing of the edge as thick round edges. In practice, we approximate the thick edges by polygons.
4) An optional step (convex) is to compute, for each hyperedge e, the convex hull $ch(e)$ of the curve associated with e. The curve of e is set to be the convex hull $ch(e)$ if only vertices that define e are included in $ch(e)$.

We assume that the reader is familiar with minimum Euclidean spanning trees and Euclidean Steiner trees (see [5]). Figure 3 is an example of such trees on a particular set of points. Given an hypergraph $H = (V, E)$, the underlying graph G, defined in the description of the PATATE method above, can be build using one of the three following methods, starting with an empty graph G:

(dummy) For each vertex v in H, a vertex $\nu(v)$ is added in G and for each hyperedge e in H, a vertex $\nu(e)$ is added in G.
Then, for each hyperedge $e = \{v_1, \ldots, v_k\}$, edges $\{\nu(e), \nu(v_1)\}$, $\ldots, \{\nu(e), \nu(v_k)\}$ are added in G. The location of $\nu(e)$ is set to be the barycenter of $v_1, \ldots, v_k$.

(span-tree) For each vertex v in H, a vertex $\nu(v)$ is added in G. For each hyperedge e in H, a minimum Euclidean spanning tree that covers the vertices that define e is computed; the edges of the spanning tree are added to G.

(Steiner tree) For each vertex v in H, a vertex $\nu(v)$ is added in G. For each hyperedge e in H, a minimum Steiner tree whose leaves are the vertices that define e is computed. A vertex is added to G for every steiner point, and the edges of the tree are added to G.

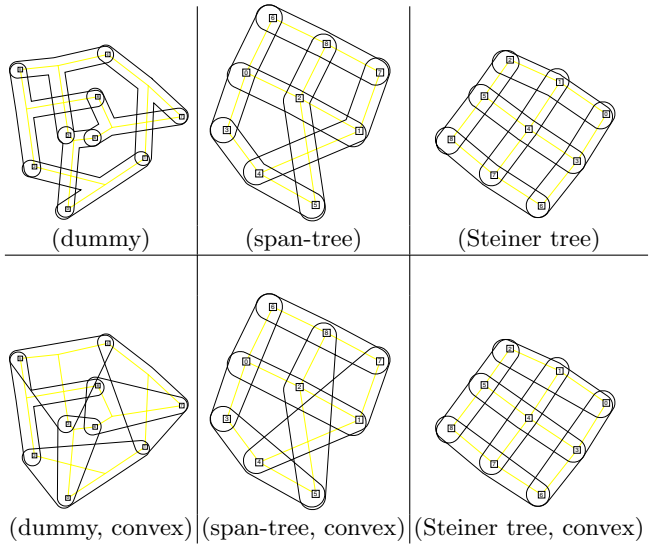

Fig. 4. Comparison between three different methods (dummy, span-tree, Steiner tree) to compute the underlying graph during the iteration steps, represented with or without the optional (convex) step.

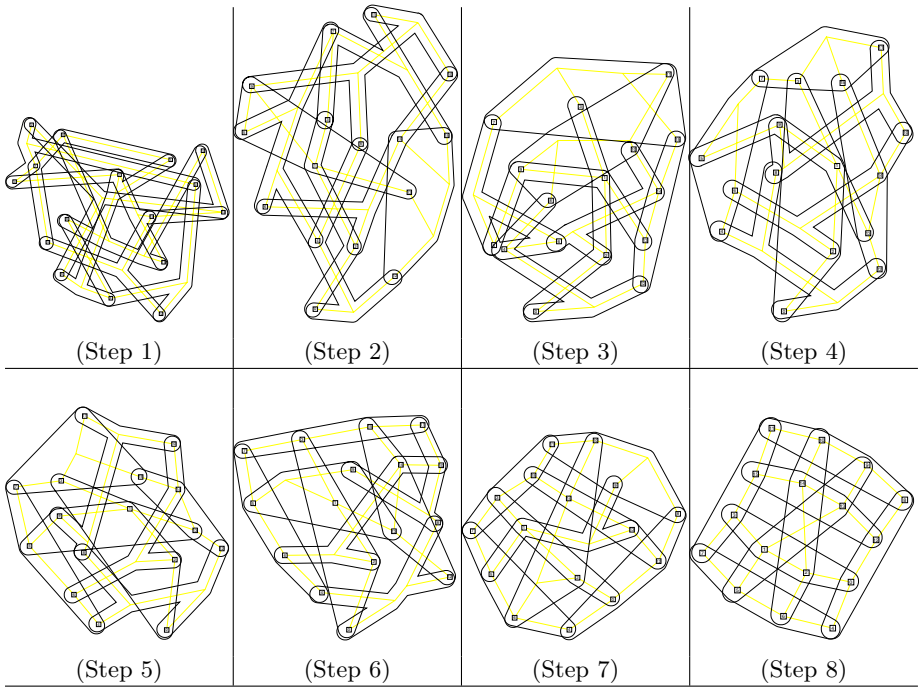

Fig. 5. Evolution of the solution at each iteration step of the PATATE method, when the Steiner and convex options are used.

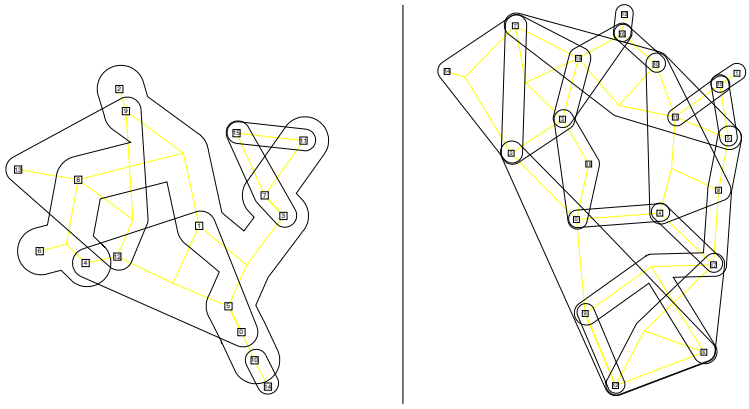

Fig. 6. Drawing of an hypergraph with a long hyperedge (left), and an hypergraph with 20 nodes, hyperedges of length at most 5, and vertices shared between at most 3 hyperedges (right).

3 Implementation

The PATATE method is implemented in C++ and uses the LEDA library [8] for the definition of the graph structures, computation of spanning trees and convex hulls. The geosteiner system [10] is used to compute Steiner trees. A modification of the force-directed algorithm by Fruchterman and Reingold [2,1] is used to compute the locations of the underlying graph G. The output of the program is a postscript file that represents the drawing of the hypergraph given as input.

Figure 4 represents drawings obtained using this three methods, and with or without the optional (convex) step. Figure 5 represents the evolution of the solution at each iteration step of the method, when the Steiner and convex options are used.

4 Conclusion

The method gives reasonable results for small hypergraphs, but can fail miserably when a vertex belongs to a high number of hyperedges (Fig. 7). The Steiner tree option can be viewed as a compromise between the (span-tree) and (dummy) options, and seems to produce the best results in practice. More experiments are needed: we did not investigate whether or not the use of Steiner trees reduces the number of iterations required to obtain a reasonable drawing. Other methods, such as the separation of two sets of points by curves with minimum perimeter, could be used instead of the simple (convex) option.

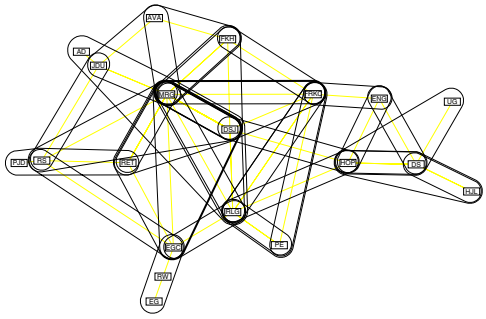

Fig. 7. Drawing of the Erdös-hypergraph from [4], using (span-tree) and (convex) options.

References

1. F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Information Processing Letters*, 74(1–2):7–13, 2000.
2. T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software-Practice and Experience*, 21(11):1129–1164, 1991.
3. D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.
4. D. S. Johnson and H. O. Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *Journal of Graph Theory*, 1987.
5. R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.
6. H. Klemetti, I. Lapinleimu, E. Mäkinen, and M. Sieranta. A programming project: Trimming the spring algorithm for drawing hypergraphs. *SIGCSE Bulletin*, 27(3):34–38, 1995.
7. E. Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 1990.
8. K. Mehlhorn and S. Naher. LEDA, a Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38(1):96–102, 1995.
9. John Venn. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 9:1–18, 1880.
10. P. Winter and M. Zachariasen. Euclidean steiner minimum trees: An improved exact algorithm. *Networks*, 30:149–166, 1997.