Drawing Graphs in Euler Diagrams

Paul Mutton[1], Peter Rodgers[1], and Jean Flower[2]

[1] Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK
{pjm2, P.J.Rodgers}@kent.ac.uk
http://www.cs.kent.ac.uk/people/rpg/pjm2/
[2] School of Computing, Mathematical and Information Sciences,
Watts Building, University of Brighton,
Lewes Road, Brighton, BN2 4GJ, UK
J.A.Flower@bton.ac.uk
http://www.cmis.brighton.ac.uk/research/vmg/people.htm

Abstract. We describe a method for drawing graph enhanced Euler diagrams using a three stage method. The first stage is to lay out the underlying Euler diagram using a multicriteria optimizing system. The second stage is to find suitable locations for nodes in the zones of the Euler diagram using a force based method. The third stage is to minimize edge crossings and total edge length by swapping the location of nodes that are in the same zone with a multicriteria hill climbing method. We show a working version of the software that draws spider diagrams. Spider diagrams represent logical expressions by superimposing graphs upon an Euler diagram. This application requires an extra step in the drawing process because the embedded graphs only convey information about the connectedness of nodes and so a spanning tree must be chosen for each maximally connected component. Similar notations to Euler diagrams enhanced with graphs are common in many applications and our method is generalizable to drawing Hypergraphs represented in the subset standard, or to drawing Higraphs where edges are restricted to connecting with only atomic nodes.

1 Introduction

The system described here links graph drawing and Euler diagram drawing into a system for drawing graph-enhanced Euler diagrams. In this sort of representation the nodes of the graph are required to appear in certain zones of the Euler diagram. Our approach is to draw the Euler diagram first, and later add the graph in a way that minimizes edge crossing and edge length.

There are various application areas which can be visualized by such structures and so benefit from the work described here such as databases [3] and file system organization [2]. However, we show our system being used with a form of constraint diagram, the spider diagram [7]. This application area is in particular need of automatic layout for the diagrams because automatic reasoning algorithms produce abstract diagrams that have no physical layout.

An Euler diagram is a collection of *contours* (drawn as simple closed curves), arranged with specific overlaps. The parts of the plane distinguished by being contained

within some contours and excluded from other contours are called *zones*. The essential structure of an Euler diagram is encapsulated by an *abstract Euler diagram*. An abstract Euler diagram is made up of information about contours and zones. Contours at the abstract level are not drawn, but have distinguishing *contour labels*. Zones are not parts of the plane, but a partition of the contour set into containing contours and excluding contours. To clarify these concepts, Figure 1 shows, first, an abstract Euler diagram, and, second, a drawn representation of the same Euler diagram. The shaded zone in the drawn diagram corresponds to the abstract zone ($\{a\},\{b\}$).

Contours : { *a*, *b* }
Zones: {({},{*a*, *b*}),({*a*},{*b*}),({*b*},{*a* })}

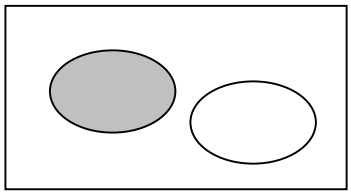


Fig. 1. The distinction between an abstract Euler diagram and a corresponding drawn Euler diagram.

The task of drawing an Euler diagram - taking an abstract diagram and producing a corresponding drawn Euler diagram is analogous to the field of graph drawing. Previous research has addressed some initial issues concerning the drawing of Euler diagrams. The paper [5] outlined well-formedness conditions on drawn diagrams and presented an algorithm to identify whether an abstract diagram was drawable subject to those conditions. If a diagram was diagnosed as drawable, then a drawing was produced. Later work, [6], sought to enhance the layout of a drawn Euler diagram using a hill-climbing approach in combination with a range of layout metrics to assess the quality of a drawing.

There has been some previous work in drawing extended graph systems. Clustered graph visualization systems are common (e.g. [4,10]), but in such structures the regions only nest and cannot intersect, hence they are not as expressive as Euler diagrams. There are a limited amount of drawing methods for more complex graph-like structures such as hypergraphs and higraphs. Hypergraphs are similar to standard graphs, but with hyperedges rather than edges. Hyperedges connect to several nodes, in contrast with standard edges which are binary as they always connect to two nodes. Hypergraphs are commonly represented in two ways: by the edge standard and the subset standard [12]. The edge standard draws hyperedges as lines, effectively adding a dummy node for each hyperedge, where the lines connecting to each node meet. Visualizing this representation reduces to a graph drawing problem. The subset standard is a representation closer to enhanced Euler diagrams, where the hyperedges are indicated by closed curves surrounding the grouped nodes. However, there are still significant differences as hypergraph closed curves that intersect have no extra meaning, and current hypergraph drawing methods [1] emphasize node groupings, putting little emphasis on the layout of the curves. Hypergraphs with binary edges are repre-

sented with the edge standard and with non binary edges represented with the subset standard are similar to commonly applied subsets of higraphs [9,11].

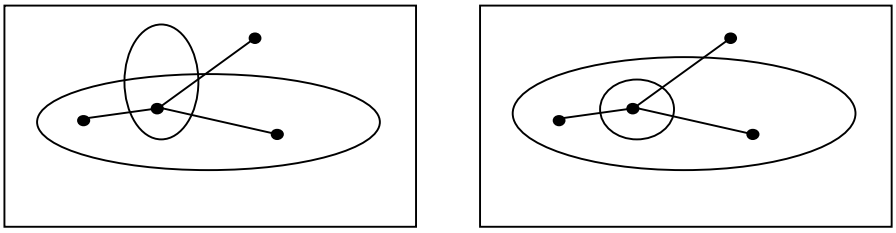


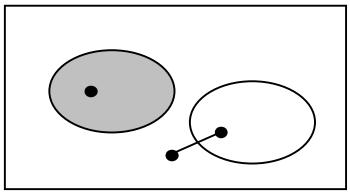
Fig. 2. Two equivalent hypergraph drawings which are different when interpreted as Euler diagrams.

In graph-enhanced Euler diagrams, the absence of a zone from the second diagram in Figure 2 would convey extra information, whereas, considered as hypergraphs, these two Figures convey the same information

Inspired by the widespread use of diagrammatic notations for modeling and specifying software systems, there has been much work recently about giving diagrammatic notations formal semantics. The analysis of a diagrammatic specification can be done using diagrammatic reasoning rules - rules to transform one diagrammatic assertion into a new diagram that represents equivalent or a weaker semantic statement.

One such notation, and reasoning system, is that of *constraint diagrams* [11]Jean, this reference is not right, what should it be? Plus, need refs for formal semantics. A simple subset of constraint diagrams, with a restricted notation and restricted rule system, is that of spider diagrams. Spider diagrams are Euler diagrams with extra notation comprising shading in zones and a graph superimposed on the diagram. The components of the superimposed graph are trees (called spiders). Contours represent sets and zones represent subsets of those sets, built from intersection and exclusion. The absence of a zone from the diagram indicates that the set corresponding to that zone is empty. Each spider drawn on the diagram has a *habitat*: the collection of zones that contain nodes of the graph. The spiders assert semantically the existence of an element in the set corresponding to its habitat. Spiders place lower bounds on the cardinality of sets. Shading in a zone (or collection of zones) indicates that the set corresponding to that zone (or zones) contains only elements for the spiders, and no more. Shading places an upper limit on the cardinality of sets. See Figure 3 for an example of a spider diagram.

Contours : { *a*, *b* }
Zones: {({},{*a*, *b*}),({*a*},{*b*}),({*b*},{*a* })}
Shading:{({},{*a*, *b*})}
Spiders : {{({},{*a*,*b*}),({*b*},{*a*})},{({*a*},{*b*})} }



Semantics:
$|A| = 1$ and $A \cap B = \{\}$ and $|U-A| \geq 1$

Fig. 3. An abstract spider diagram and a corresponding drawn spider diagram.

The semantics of spider diagrams provide a foundation upon which we build reasoning rules. In the case of spider diagrams, there are seven rules which transform a spider diagram into another. For example, one rule transforms a diagram with an absent zone into the equivalent diagram which contains the zone and it is shaded. This reasoning rule changes the structure of the underlying Euler diagram and necessitates reconstruction of a drawn diagram. A sequence of reasoning rules, applied to a premise diagram, gives a proof which ends with a conclusion diagram. An example of such a proof is shown, drawn by hand, in Figure 4.

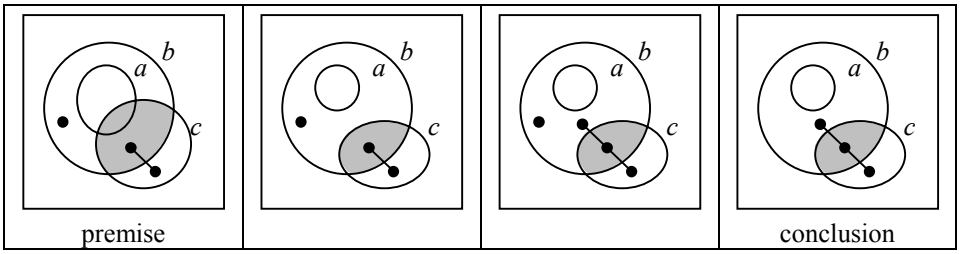


Fig. 4. An example of a proof in the spider diagram reasoning system

The full spider diagram reasoning system allows for the manipulation and interpretation of compound spider diagrams: that is, expressions built up from spider diagrams using the propositional logic connectives "and" and "or". This extension leads to many more reasoning rules, giving a sound and complete reasoning system, equivalent in its expressiveness to monadic first order predicate logic with equality. Detailed descriptions of the system, its rules and its expressiveness can be found in [12]Is this reference right?.

We have developed a tool to assist users with the application of reasoning rules to transform diagrams. At the heart of this must be an algorithm to generate diagrams for presentation to the user as the outcome of a rule application.

2 Drawing Euler Diagrams Enhanced With Graphs

In this section we describe our three stage generic method for laying out graphs on Euler diagrams. The software system has been implemented in Java.

2.1 Stage 1: Euler Diagram Smoothing

The basic process of drawing Euler diagrams in stage 1 has been detailed previously [6]. In outline, firstly we produce an initial diagram based on the zone specification as described in [5]. This results in a structurally correct, but not very well laid out diagram. We then apply a multicriteria optimizer, which attempts to improve a weighted

sum of various diagram layout criteria using a hill climbing method. This adjusts the contours by both moving them and moving the individual points of the polygons that are used to represent them. It assesses the layout formed on each single move for the presence of the correct zones and to see if the change has improved the weighted sum. We use several criteria for measuring diagram features, such as contour smoothness, contour size, zone area and contour closeness. The criteria and the hill climber are described in [6].

This system has since been extended to deal with nested diagrams. Nested Euler diagrams have subdiagrams entirely enclosed in a zone of a containing diagram. To draw a nested diagram, assuming we have a mechanism for drawing each atomic (non-nested) part independently, the first step is to identify, in the abstract diagram, which are the atomic components and which zones of containing diagrams each nested part belongs to. Each atomic component can be drawn and this tree-structure of drawn atomic components is combined into a single diagram as follows. For each zone which contains sub-diagrams, find its bounding box and consider sequences of sub-boxes within the bounding box. The sub-boxes occupy a fraction i/j of the bounding box, and are placed sequentially at $(j-i)^2$ positions scanning the whole bounding box (starting centrally). As j gets larger, the subboxes shrink and eventually one will be found which fits inside the zone. This sub-box is partitioned into disjoint boxes, within which the nested diagrams are inserted. This process is illustrated in Figure 5.

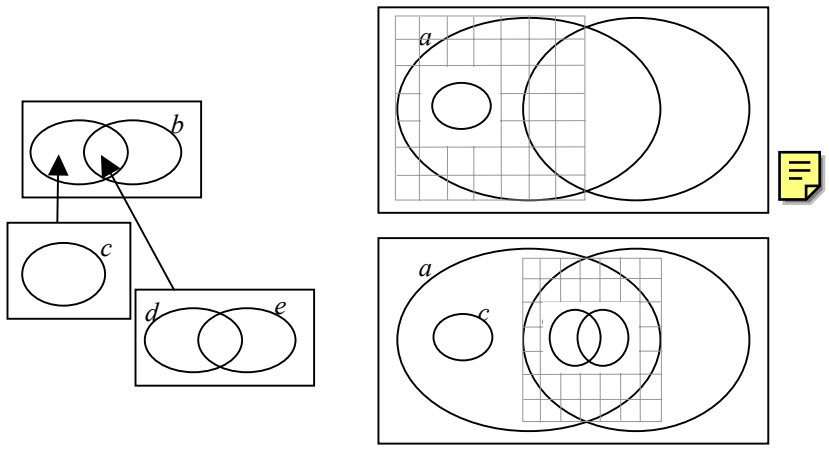


Fig. 5. Nesting Euler diagrams.

Once the nested diagram has been built in this way, the next step is to improve its appearance by smoothing. As the nesting can be arbitrarily deep the amount of movement of polygons and polygon corners could be too large for very small nested contours. Hence, the amount of movement has been scaled to be proportional to the size of the contour (in fact the bounding box of the contour) against the size of the whole diagram.

The result of Stage 1 is normally a well laid out Euler diagram. The graph can then be superimposed as described in the following sections.

2.2 Stage 2: Finding Locations for Nodes

A node belonging to a particular zone must be placed such that the node is contained within the region defined by the drawn zone. Each concrete zone is defined by a sequence of line segments. We do not concern ourselves with disconnected zone areas, as these are not present in a well-formed [5] Euler diagram, however, for nested diagrams, at least one zone fails to be simply-connected (i.e. it's ring-shaped, or worse; see Figure 6). Zones which are simply connected (i.e. disc-like) have one polygon as their boundary, but non-simply connected zones have multiple polygons bounding them.

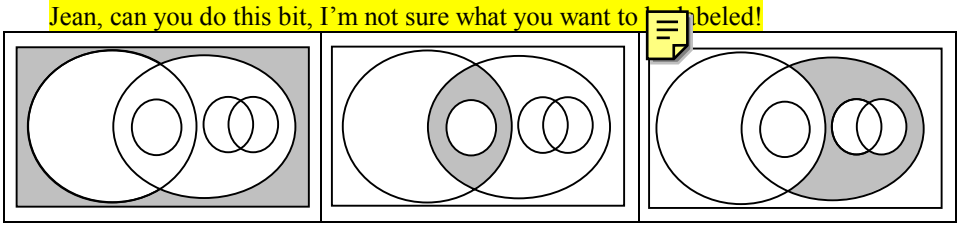


Fig. 6. Three examples of non-simply-connected zones

A variety of possible strategies exist for the initial placement of a node inside its containing zone. We use a fast and simple method that is primarily concerned with ensuring that the node is contained inside the zone, regardless of how bad that placement is. Subsequent application of a force model refines the placement so that the node is not too close to any of the boundaries of the zone. The force model also ensures that all nodes sharing the same zone are reasonably spaced.

The initial placement of a node requires a line to be drawn through the containing zone. For simplicity of implementation, this line is horizontal and passes through the bounding box of the concrete zone. The y-coordinate of the horizontal line is chosen randomly between the range of the bounding box in order to give a scattering effect when there is more than one node present in a zone. By intersecting the bounding box horizontally, we can be certain that there is at least one subinterval of the line that is contained by the area of the concrete zone.

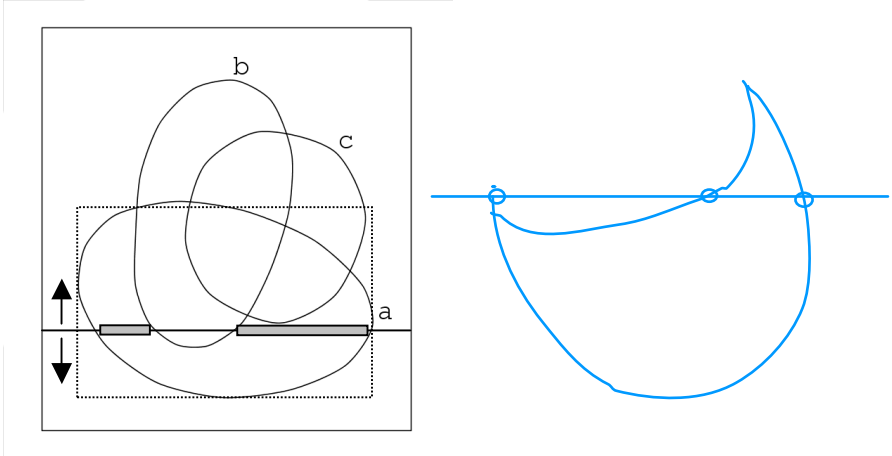


Fig. 7. Candidate locations for a new spider foot in zone *a* excluding *b*,*c*. The horizontal line is placed such that it intersects the bounding box of zone *a* at a random height. This diagram shows two subintervals where it is valid to place the new spider foot.

An ordered set is built up from the intersection points of the horizontal line and the line segments which make up the boundary of the zone. This set must contain at least two points, and any location between the 2n-1th and 2nth intersection point must belong to the zone.

The Stage 1 method for placing nested diagrams described in Section 2.1 could have been used for the initial placement of nodes. However this node placement method is faster as we are placing a point rather than a shape with a bounding area and we are unconcerned about a central placing of the point, anticipating the refinement which is described next.

After initial placement, refinement of node locations is achieved by applying a force model to the set F of nodes in the zone. We introduce a repulsive force acting between each pair of nodes in the zone, causing them to become evenly distributed. This repulsive force is inversely proportional to the separation d, and proportional to the number of nodes, $|F|$, in the zone. A constant c is used to affect the desired separation between pairs of nodes. This repulsive force is based on the force model by Fruchterman and Reingold [15] and is commonly used in force directed graph layout.

$$\text{Repulsive force between two nodes} = |F| \times \frac{c}{d}.$$

To prevent nodes from escaping from a zone or getting undesirably close to the boundary of a zone, we make each line segment in the zone exert a repulsive force on each contained node. It is desirable to let the set of nodes spread about a reasonably large area of the zone, however it is still essential to keep each nodes away from the boundaries of the zone. For this reason, we depart from the previously used force model and make the repulsive force acting on a node proportional to the inverse square of the distance from the line segment. This encourages nodes to spread over a reasonable area with very little chance of getting too close to a boundary due to the prohibitively high resultant forces.

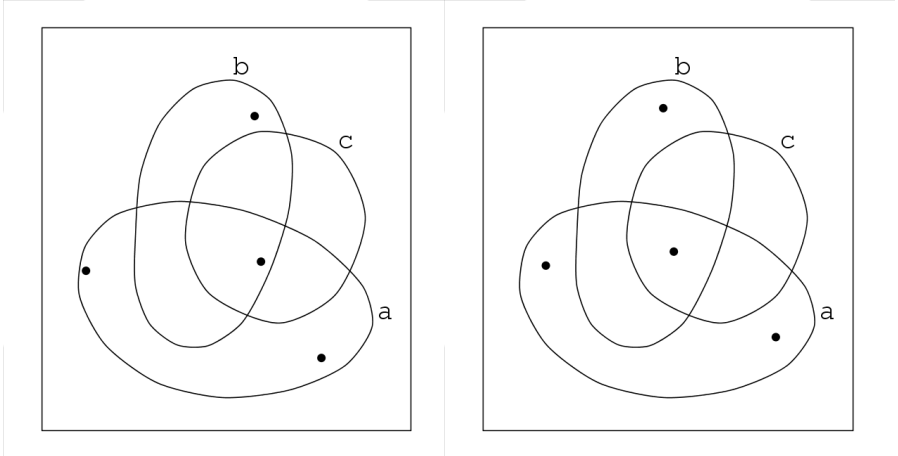


Fig. 8. Initial placement of spider feet (left) and refinement under the force model (right).

The repulsive force is proportional to $|F|^2$, as this helps to contain larger sets of nodes. As the zone may consist of an arbitrary number of line segments of arbitrary lengths, the repulsive force is also proportional to each length.

$$\text{Repulsive force between a line segment and a node} = |F|^2 \times \frac{lc}{d^2}.$$

We have observed that better results can be obtained when there are more line segments bounding a zone. We use a method that breaks a zone boundary into more line segments without affecting the region contained; typically so there become more than a hundred new line segments. This is done by dividing each existing line segment into two new line segments of equal length. The process is repeated until it yields enough new line segments. This reduces the chance of a node escaping from a corner of the zone when the force model is applied.

The simulation of the force model is an iterative process. For each iteration, the resultant force acting on each node is the sum of all repulsive forces from the line segments of the containing zone and the repulsive forces from all other nodes in the same zone. After calculating all of the resultant forces, the location of each node is updated by moving it a small distance in the direction of the force. The distance of the movement is proportional to the magnitude of the force. After a number of iterations, the system nears an equilibrium and the nodes occupy their new locations.

2.3 Stage 3: Placing Nodes in Node Locations

The previous stage calculates locations for nodes. We can think of these locations as being candidate locations for the set of nodes in the zone, and we are free to swap the location of pairs of nodes, within a zone, without changing the meaning of the diagram (see Figure 11). We use a simple hill climbing approach on this with two metrics to improve the quality of the diagram.

One desirable feature of a diagram is to have a minimal number of edge crossings. Our first metric returns the number of edge crossings in the current diagram, so values closer to zero will represent a better quality of layout in terms of edge crossings. To further enhance the understandability of the diagram, we introduce a second metric, which is based on the total length of all edges in the diagram. Shorter edges make graphs easier to navigate and identify, so the value returned by this metric will represent an improvement in the layout if the value is closer to zero.

In our current system, we are only concerned with simple straight-line edges, although it is worth noting that our software can deal with non-simple edges. For example, some notations use curves or shapes to represent special edges and our system is able to detect intersections with these more nonlinear edges.

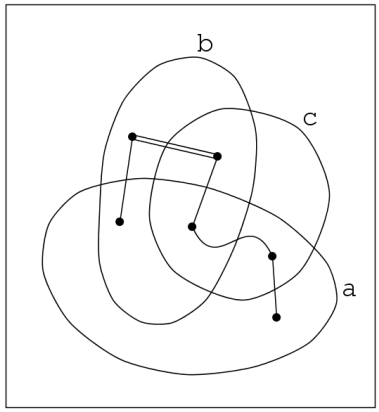


Fig. 9. A diagram demonstrating the different types of edges that are supported by our system. Intersections with the more complicated types of edges can still be computed.

As the value returned by the edge length metric is based on the sum of edge lengths in the diagram, we make this value dimensionless by dividing it by the square root of the area of the diagram. This makes the metric return the same value for a particular diagram, regardless of the scaling.

The two metrics are combined as a weighted sum to work out the current quality of a diagram. As we have determined minimization of edge crossings to be the most important factor, we apply a much higher weighting to this metric. That is, we are unlikely to reduce the total edge length in a diagram at the expense of introducing a new edge crossing.

In our implementation of the system, we use a weighting of 1 for the edge crossing metric. The weighting of the edge length metric is relative to this and is chosen such that when the returned value is multiplied by the weighting, the value is typically less than 1. Larger values may allow total edge length to be reduced at the expense of introducing new edge crossings. Our implementation uses a weighting of 0.01 for the edge length metric weighting.

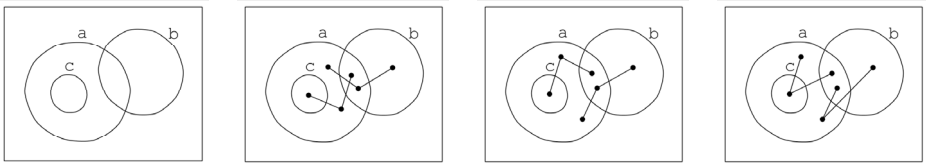


Fig. 10. Total quality metrics for some graph enhanced diagrams.

The hill climber is also an iterative process and runs for either a fixed number of iterations, or a user may interact with the process and apply more iterations if it is deemed necessary. Each iteration begins with selecting a random zone that contains more than one node. A random pair of nodes is selected from this zone and their locations are swapped. This does not alter the meaning of the diagram, as they both lie within the same zone. If the new quality of the diagram is worse than before, the nodes are swapped back to their original locations; otherwise, the change is kept. After a number of iterations, the quality of the diagram according to the metrics improves.

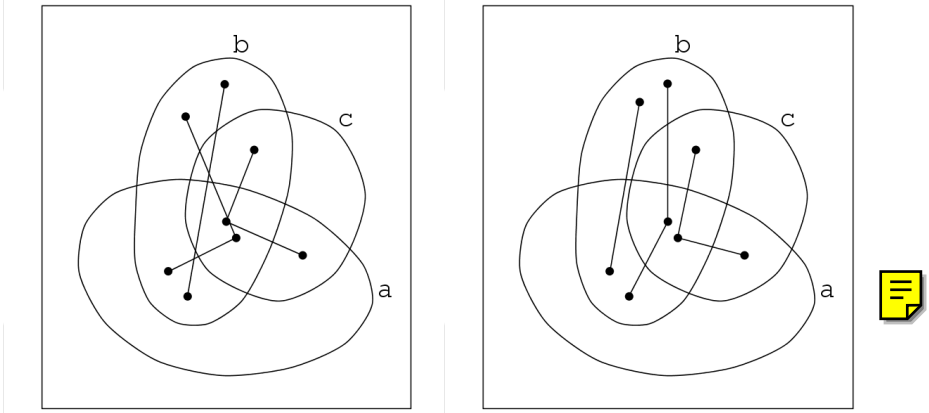


Fig. 11. A diagram with 4 edge crossings (left) and the same diagram produced using the hill climber, with no edge crossings (right). Notice the common locations for all nodes. The right hand diagram has had 3 pairs of nodes swapped, in zones *b*, *ab* and *abc*.

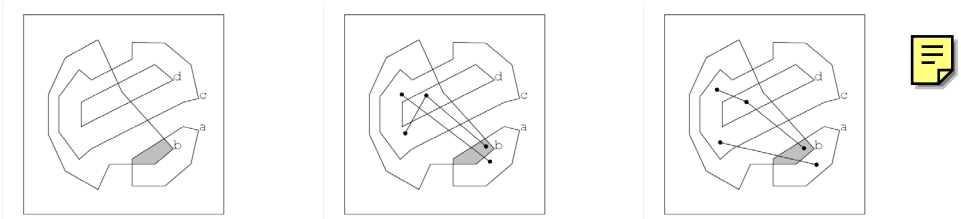
3 Drawing Spider Diagrams Enhanced With Graphs

In this section we describe how we apply our method to spider diagrams. The method is essentially that described in Section 2, except that spider diagrams do not use arbitrary graphs connecting nodes, instead nodes are connected in spanning trees, and the manner in which the nodes are connected in the spanning tree is not significant. The abstract syntax of spider diagrams expresses spiders purely in terms of their habitat. A spider whose habitat comprises three zones, z_1, z_2, z_3 can be drawn with a

graph edges (the spider's *leg*) drawn between graph nodes (the spider's *feet*) in z_1 and z_2 and a second leg between graph nodes (the spider's *feet*) in z_2 and $z3$. An alternative drawing might draw legs between z_1 and z_2 and between z_1 and z_3. Only once a spider is drawn do we know which of its feet have a leg between them. As we only have the information about which sets of nodes are connected, our drawing method needs an additional process that develops a spanning tree between the nodes.

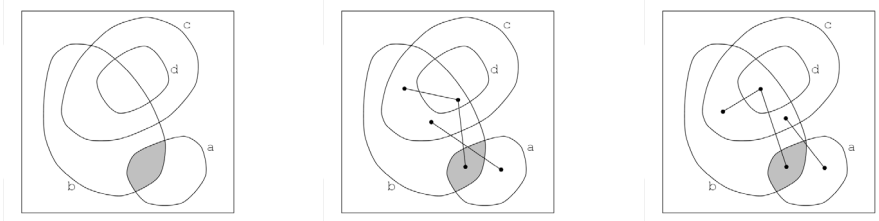
Once the feet for each spider have been placed, it is possible to use Prim's or Kruskal's algorithm to form a minimal spanning tree. This completes the concrete representation of the spider with the smallest total edge length, but does not take into account edge crossings. As our hill climbing method gives preference to changes that reduce edge crossings, we do not create a minimal spanning tree, but trivially form a chain of spider legs that connect each spider foot.

A proof in the spider diagram reasoning system can be elicited from a user, with a software tool assisting in the application of reasoning rules, or, proofs can be automatically generated between given premise and conclusion diagrams [two refs here]. A proof is essentially a sequence of diagrams with descriptions of rule applications obtainable between adjacent diagrams. An example is shown in Figures n, n+1 and n+2, where the rules "Add Shaded Zone" and "Add Spider Foot" have been applied. The first rule changes the underlying Euler diagram, and the second rule changes the superimposed graph. Without any results on drawing spider diagrams, the proof can only be presented in its abstract form (Figure n). The preliminary work on drawing can present the proof with correct but unappealing diagrams (Figure n+1). After applying the algorithm described in this paper, the proof is presented in a most readable fashion (Figure n+2). Jean, I'm not too sure what diagrams you wanted me to use for the figures mentioned in this paragraph. Let me know and I'll make them (paul).



Initial Euler diagram | Initial node placement | Refined node placement and crossing reduction

Fig. 12. Embedding graphs into an Euler diagram that has been drawn automatically.



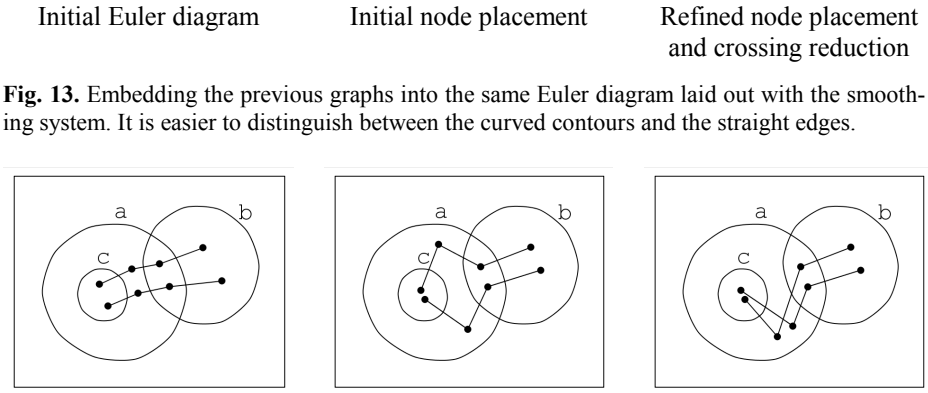


Fig. 13. Embedding the previous graphs into the same Euler diagram laid out with the smoothing system. It is easier to distinguish between the curved contours and the straight edges.

Fig. 14. Different layouts for the same graphs in an Euler diagram. The bad automatic layout occurs when both nodes in zone *a* are initially placed close to each other and reach a local minima while the force model is being simulated. In this case, it is not possible to reduce edge crossings without moving nodes – a task that could be performed by simulated annealing.

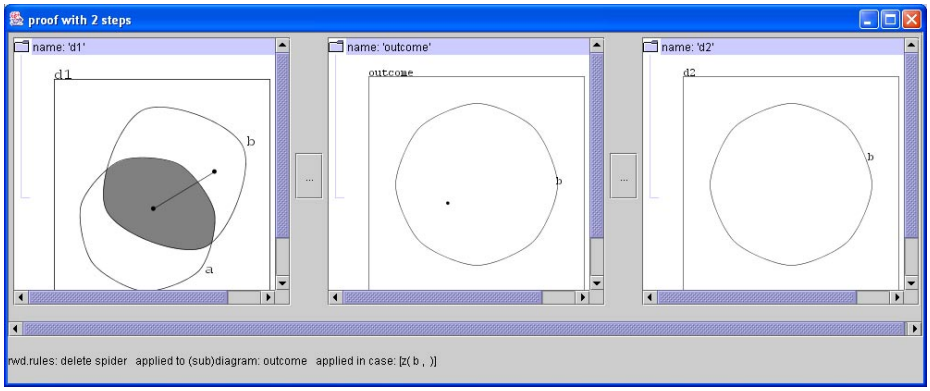


Fig. 15. Using our system to draw each step of an automated proof.

4 Conclusions and Further Work

The example of a proof shown in Figure n was chosen well, to ensure that all the component diagrams are drawable (as described in [5]). More work needs to be done to resolve, and draw, diagrams that are currently diagnosed as "undrawable".

Another question raised by the proof-presentation application is that of "continuity of proofs". When one small change is made (a new zone or a new spider foot), the proof would be best understood if the drawn diagram closely resembles the preceding diagram, except for a "local change". This task generalises to that of drawing one diagram given a "context" - another diagram which is structurally similar, or even

given a context which is a library of already drawn examples. If all of a set of diagrams are provided only in their abstract form, how can the diagrams be generated to maximise the likelihood that readers can see similarities between the diagrams?

At the moment, the optimization of the graph layout relies on swapping nodes that are in the same zone. A further addition to this for spider diagrams is to change the spanning tree of a spider as a move in the hill climber, in an attempt to improve edge crossings and edge length of the final graph. We feel that this can improve the layout of spiders.

It would be desirable when drawing diagrams such as proof sequences to maintain the mental map between the dynamic visualizations of each proof step. There are several possible ways to achieve this. One method is to include a mental map criteria across all diagrams when performing hill climbing at both the Euler Diagram and node location stages. Another method is to draw the first diagram nicely, and then attempt to draw subsequent diagrams incrementally to remain as close to previous ones as possible.

References

1. François Bertault, Peter Eades. Drawing Hypergraphs in the Subset Standard. GD 2000. LNCS 1984.164-169.
2. R. De Chiara, U Erra and V. Scarano. VENNFS: A Venn-Diagram File Manager. Proc. IEEE Information Visualization (IV03). pp. 120-126. 2003.
3. M.P. Consens and A.O. Mendelzon. Hy+: A Hygraph-based Query and Visualization System. In Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, pages 511-516, 1993.
4. Peter Eades, Qingwen Feng. Multilevel Visualization of Clustered Graphs. GD96. LNCS 1190. pp. 101-112.
5. J. Flower and J. Howse. Generating Euler Diagrams, Proc. Diagrams 2002, Springer Verlag, pp. 61-75. 2002.
6. J. Flower, P. Rodgers and P. Mutton. Layout Metrics for Euler Diagrams. Proc. IEEE Information Visualization (IV03). pp. 272-280. 2003.
7. J. Flower and G. Stapleton. Automated Theorem Proving with Spider Diagrams, submitted to Computing Australasian Theory Symposium (CATS04).
8. J. Flower, J. Masthoff and G. Stapleton. Generating Readable Proofs : A Heuristic Approach to Theorem Proving With Spider Diagrams, submitted to Diagrams '04.
9. David Harel. On Visual Formalisms. Communications of the ACM. 31(5). pp. 514-530. 1988.
10. David Harel, Gregory Yashchin. An Algorithm for Blob Hierarchy Layout. Working Conference on Advanced Visual Interfaces. pp. 29-40, May 2000.
11. Higraph web page: http://db.uwaterloo.ca /~gweddell/higraph/higraph.html.
12. Erkki Mäkinen, How to draw a hypergraph. International Journal of Computer Mathematics 34 (1990), 177-185
13. Higraph web page: http://db.uwaterloo.ca /~gweddell/higraph/higraph.html.
14. Erkki Mäkinen, How to draw a hypergraph. International Journal of Computer Mathematics 34 (1990), 177-185
15. T.M.J. Fruchterman, E.M. Reingold. Graph Drawing by Force-directed Placement. Software – Practice and Experience Vol 21(11). pp. 1129-1164. 1991.