Embedding Wellformed Euler Diagrams

Peter Rodgers[1], Leishi Zhang[1], Gem Stapleton[2], Andrew Fish[2]
1. University of Kent, UK, 2. University of Brighton, UK
{P.J.Rodgers@kent.ac.uk, L.Zhang@kent.ac.uk, G.E.Stapleton@brighton.ac.uk,
Andrew.Fish@brighton.ac.uk}

Abstract

Euler diagrams are collections of labelled closed curves. They are often used to represent information about the relationship between sets and, as such, they have numerous applications including: visualizing biological data, diagrammatic logics, and visual database querying. Various methods to automatically generate Euler diagrams have been proposed recently. Typically, the generation process starts with an abstract description of an Euler diagram, which is then converted to a planar dual graph. Finally, the process attempts to embed the Euler diagram from the dual graph. This paper describes a method for embedding wellformed Euler diagrams from dual graphs. There are several mechanisms to generate dual graphs but, prior to the novel work described here, no general method for embedding a wellformed Euler diagram from a dual graph had been demonstrated. The method in this paper achieves an embedding of any wellformed Euler diagram. The method first triangulates the dual graph. Then, using the faces of the triangulated graph, an edge labelling technique identifies the vertices of polygons which form the closed curves of the Euler diagram. The method is demonstrated by a Java implementation. In addition, this paper discusses a number of layout improvements that can be explored for this embedding method.

Keywords---**Euler diagrams, Venn diagrams, graph drawing, information visualization.**

1. Introduction

Euler diagrams are a popular and intuitive notation for representing information about sets and their relationships. To illustrate, the Euler diagram in Figure 1 shows the relationship between parts of the British Isles. Euler diagrams allow the representation of concepts such as one set being a subset of another or that two sets are disjoint. The term Venn diagram is often applied to such examples, however Venn diagrams are a specific type of Euler diagram which represents all set intersections [12], and, as the number of sets increases, they quickly becomes visually cluttered.

Euler diagrams consist of a finite collection of labelled closed curves, called contours. The minimal (non-empty) regions are called zones. For example, Figure 1 has a contour with label "Ireland", and a contour with label "United Kingdom". The zone which is inside both these contours contains the item "Northern Ireland". The zone that is in the contour "Ireland" but not in the contour "United Kingdom" contains the item "Republic of Ireland".

The demonstrable popularity of Euler diagrams lies in their wide-ranging applications, including the visualization of statistical data [1], displaying the results of database queries [17] and representing non-hierarchical computer file systems [2]. They have been used in a visual semantic web editing environment [16] and for viewing clusters which contain concepts from multiple ontologies [8]. Another major application area is that of logical reasoning [15] and such logics are used for formal object oriented specification [10].

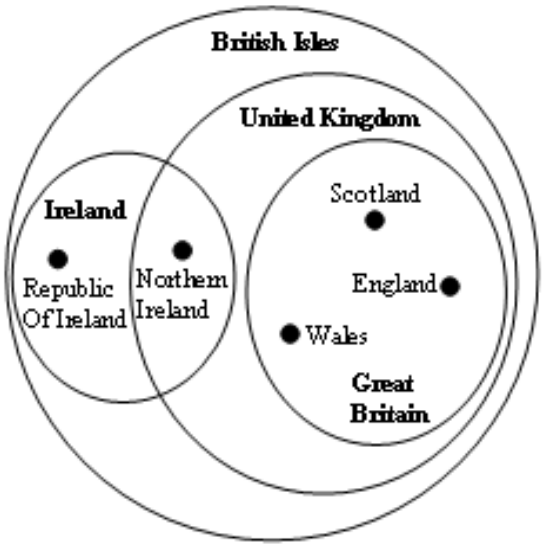

Figure 1. British Isles Euler diagram by Sam Hughes.

Currently, in all but some restricted cases, Euler diagrams must be laid out by hand. In all of the above application areas, the automated layout of Euler diagrams will bring substantial benefits, allowing complex diagrams

can be developed and permitting Euler diagrams to be used with visualization systems.

When defining generation and layout techniques, one should place a high importance on the usability of the diagram produced. Usability can be correlated with a range of desirable properties, sometimes called wellformedness conditions. Perhaps the most commonly required properties are the absence of concurrency between curves, the absence of triple points (or worse) of intersection, the absence of disconnected zones, and the use of simple curves only. Each of the diagrams in Figure 2 breaks one or more of the wellformedness conditions.

In this paper we use single letters to label contours. Each zone can be described by the contour labels in which the zone is contained. An abstract description of an Euler diagram, describing precisely which zones are required to be present, is a collection of zone descriptions. For example, the abstract description for the Euler diagram in Figure 3c is ∅ **a b ab bc abc**, where ∅ indicates the zone which is contained by no contours, called the *outside zone*.

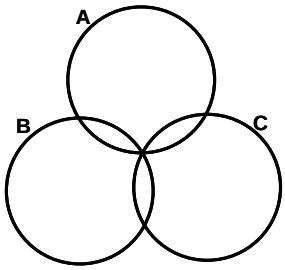

Figure 2a.
A triple point.

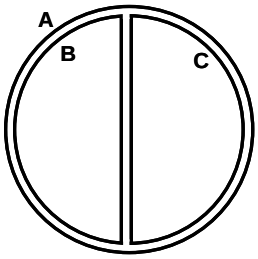

Figure 2b.
Concurrent curves.

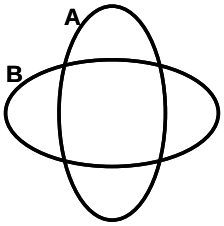

Figure 2c.
Disconnected zones.

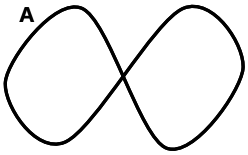

Figure 2d.
A non-simple curve.

Previous work on generation produces Euler diagrams satisfying certain wellformedness conditions [1],[4], [11],[17]. However, abstract descriptions that cannot be drawn under certain wellformedness conditions exist, such as ∅ **a b c ab ac bc** (shown embedded in Figure 2a) and ∅ **ab bc** (shown embedded in Figure 2b). In addition there are nine set abstract descriptions that cannot be drawn with simple curves [17].

To generate an Euler diagram, the standard approach is to take an abstract description and, first, derive a planar dual graph. Each node in the dual graph represents a zone and is labelled by that zone's abstract description. In simple cases, edges between nodes arise when there is exactly one contour label difference between node labels; this single label is used to label the edge. The dual graph is then embedded in the plane (see Figure 3a); see [4] for further details. Finally one attempts to embed the Euler diagram by routing each contour with label *l* through the edges labelled *l* (see Figure 3b). In these diagrams, the dual graph is shown with dotted lines to more easily distinguish it from the Euler diagram.

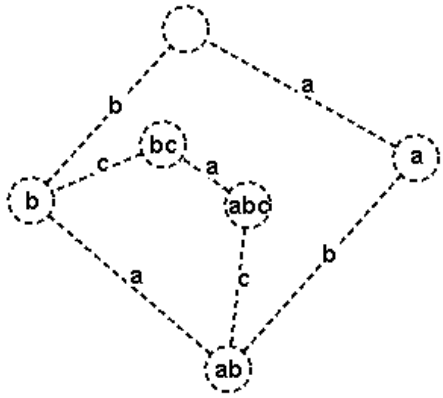

Figure 3a. A dual graph for ∅ a b ab bc abc.

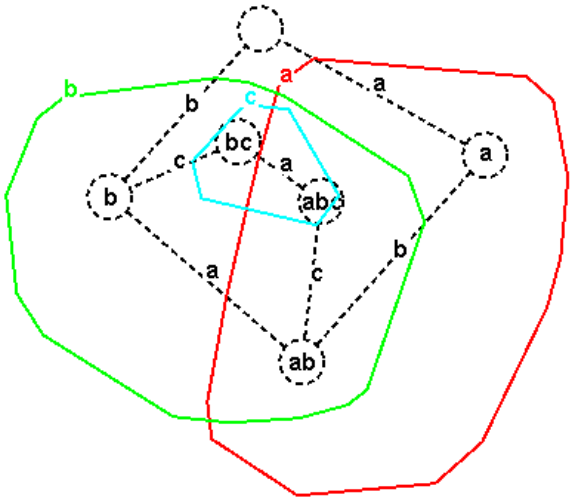

Figure 3b. An embedding of the Euler diagram from the dual.

Even if an embedding can found, there is no guarantee that the layout of the Euler diagram will be usable. Apart from the properties illustrated in Figure 2, other features such as smooth, well spaced contours are desirable. The layout of the dual graph will have an impact on the presence of features, as will the mechanism used for determining the routing of the contours through the dual. An Euler diagram after layout improvement, is shown in Figure 3c.

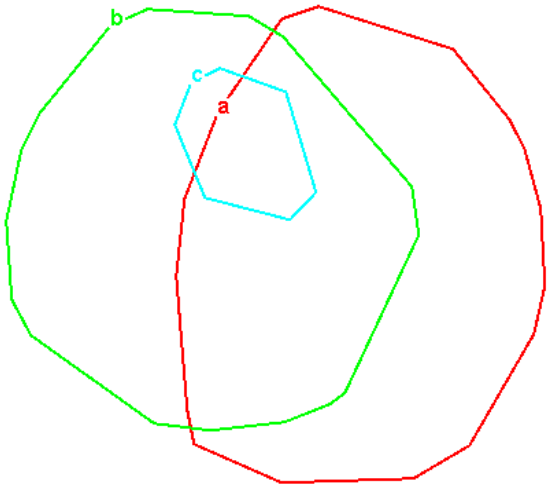

Figure 3c. The final diagram ∅ a b ab bc abc**.**

This paper addresses the Euler diagram embedding problem. We define a technique for routing contours through the dual in such a way that we create a wellformed diagram (a diagram that has contours which meet transversely whenever they intersect and has none of the properties shown in Figure 2: no triple points, concurrent curves, disconnected zones or non-simple curves) whenever this is theoretically possible. We take a dual generated by the method described in [4] that is guaranteed to admit a wellformed Euler diagram and triangulate it. We then label the triangulation edges with the difference between incident node labels; these labels identify the contours that will pass through the edge. The labels are then placed in an order on the edge which indicates how the contours will pass through each triangle, that is, which (if any) contours will intersect in the triangle.

This paper also presents methods for improving the layout of the diagrams which take advantage of this triangulation approach. First, we modify the plane embedding of the dual through a force directed layout method, which adds a node-edge repulsive force in addition to the normal node-node repulsive force. Secondly, we optimize the positioning of the edge labels (each of which is placed where the relevant contours will cut the edge) in order to obtain contours with a more regular shape.

The method we use to route the contours through triangulated duals is described in Section 2. Section 3 discusses improvements to the layout of Euler diagrams. Section 4 shows some examples of the method in operation. Finally, Section 5 gives our conclusions and further work.

2. Forming diagrams from dual graphs

This section describes the process of taking a plane embedding of a graph dual in order to produce an embedding an Euler diagram with the required zone set.

2.1. Generating a wellformed dual graph

To generate a dual graph we follow the methodology of [4]. This generates connected duals that admit diagrams that have contours which, whenever they intersect, do so transversely. There are at most two contours intersecting at any point, with no concurrent contours, all contours are simple curves, and each zone is a connected component of the plane. Starting with an abstract description, the method first creates the nodes of a dual graph called the superdual; the zone descriptions are the node set and edges connect two nodes whenever they have exactly one contour label difference; and this difference forms the edge label. The superdual is then transformed (possibly including edge removal) into a connected plane graph, called the plane dual, that satisfies the connectivity conditions and face conditions (defined below). Currently implementation imports the open source library JGraphEd to test for planarity and to find a plane layout of the dual, although we are experimenting with other planar layout tools.

The connectivity conditions are a simplification of those used in [5] and state that the dual is connected and for each contour label used in the abstract description, if the nodes without that contour label present are removed (recall, a node is a collection of contour labels) then the graph remains connected and, similarly, if the nodes without that contour label present are removed then the graph must also remain connected.

In order to define the face conditions, we first define crossing index: the crossing index of a face is the number of pairs of labels which occur on the edges which bound the face in a non-nested manner. For example, given the face edge label cycle **uavbwbxay** the letters **a** and **b** are nested whereas they are not nested in **uavbwaxby.** The non-nestedness of a pair of labels corresponds to the need for their associated contours to cross within the face. The face conditions state that for each face of the plane dual graph the crossing index must be one less than the number of distinct labels on the edges bounding the face.

The embedding method presented in this paper takes a dual that meets these conditions, and draws the contours around the nodes respecting the enclosure information provided. So, each contour will enclose precisely the nodes which include that contour's label. The contours are drawn with polygons and we identify which faces that each contour must pass through by considering the edge labelling: if a label appears on an edge bounding a face then the contour with that label must pass through the face. Moreover, the contour cuts through precisely the edges with that contour label.

2.2. Difficulties when routing contours

In general, straight lines cannot simply be drawn between edges of the dual to show where contours pass through faces, because a face may not be convex. Hence

the line could cut other edges which do not include the contour label, possibly introducing incorrect contour intersections and so failing to form the required zone set. If an arbitrary polyline routing through the face is taken, incorrect intersections can again occur, also possibly failing to form the required zone set. See Figure 4, where the zone **c** appears but does not exist in the abstract description, and the zone **a** is a disconnected region, appearing both at the bottom and top right of Figure 4. The difficulty of routing contours motivates the use of a triangulation. The convex nature of the triangles means that the above problems can be avoided, but we must establish how to route contours through the triangles.

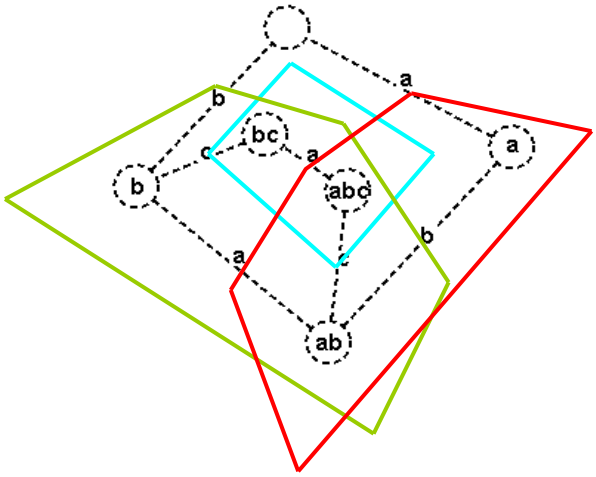

Figure 4. An incorrect embedding for ∅ a b ab ac abc**.**

2.3. Embedding a Euler diagram

First, we triangulate the faces of the plane dual graph. Instead of triangulating the infinite face we form a border of nodes with empty labels around the graph and triangulate the polygon that is formed. As with the dual graph, each triangulation edge is labelled with the difference between the labels present in its incident nodes, see Figure 5a, the triangulated edges are shown in blue and with a wider dash than the dual edges. Again, as with the dual, the labels on the triangulation edges indicate which contours will cut them when we produce an embedding. However, unlike the dual, there may be multiple contours passing through any triangulation edge.

Figure 5a also shows the label ordering along the triangulation edges identifying where each contour will pass through the edge. The ordering is produced in such as was as to ensure that all required contour crossings occur exactly once in a dual face (the method is described below). Figure 5b shows the contours passing through the subsequent contour cutting points. For the purposes of this method we treat dual graph edges as triangulation edges.

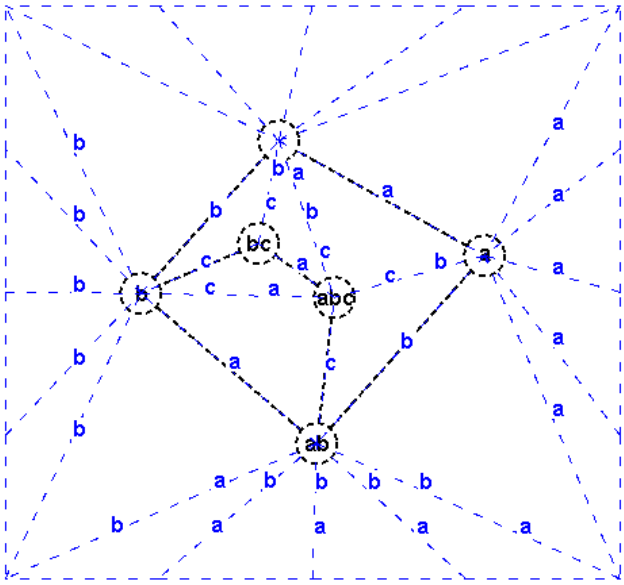

Figure 5a. A triangulated graph. The 16 border nodes around the bounding rectangle are hidden.

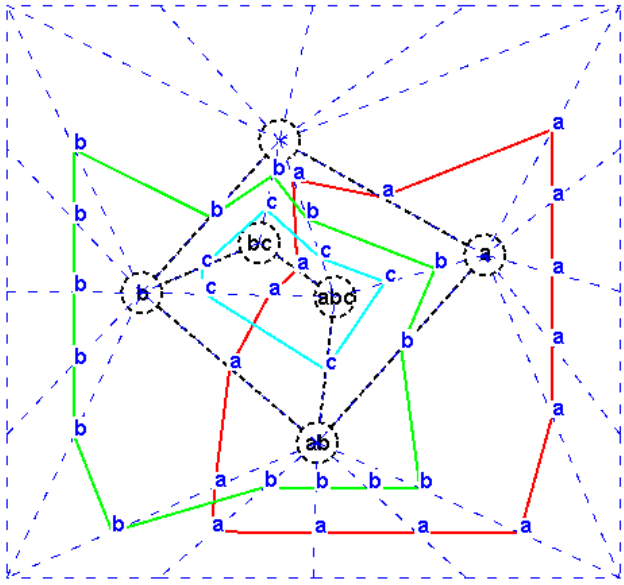

Figure 5b. Embedding contours through the triangulated dual.

The contours that cross in a dual face can be found by the method described in Section 2.1, i.e. through establishing nestededness in the face-cycle around the face. In the large *dual* face in Figure 6a with face-cycle **abcacb** the contours **a** and **b** must cross, as must **a** and **c**. It is then possible to assign these crossings so that they occur in a particular face of the triangulated dual, provided those contours pass through the triangle.

The triangulation edge label ordering process progresses through the face, assigning the order of the labels on a triangulation edge in a triangle where two

triangulation edges already have their contour order assigned. At the start of the process there are at least two triangles with two edges assigned labels; these are triangles with two edges that appear in the original face - because they are labelled with a single contour they have a trivial contour order. Any ordering of the labels is permitted provided the resulting ordering around the triangle results in the enclosed face satisfying the face conditions. Figure 6a shows triangulation edges for the inner faces which have not yet got ordered labels. The two triangles connecting nodes **∅, b, bc** and **a, ab, abc** have two edges with contour order assigned. If the triangle made from nodes **∅, b, bc** has a label order given to its third edge as shown in Figure 6b, then a further triangle (made from nodes **∅, b, bc**) will have two edges with a label order. The process then continues until all triangulation edges are fully assigned, as in Figure 5a.

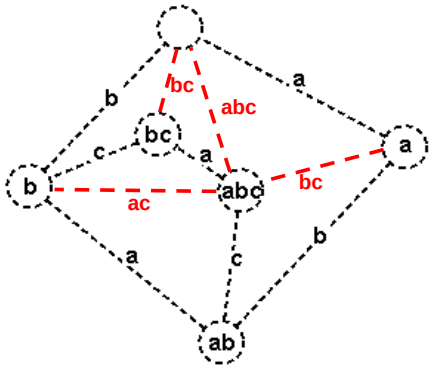

Figure 6a. Unassigned triangulation edges

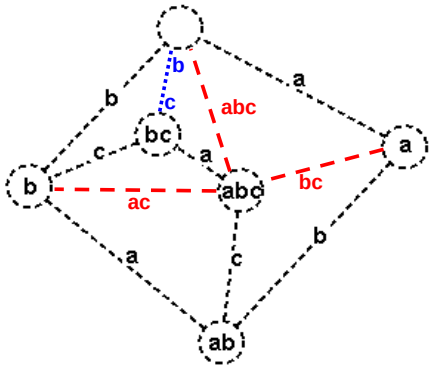

Figure 6b. Ordering edge labels

Contours cross triangles with straight lines, so the order of labels on triangulation edges defines which contours will cross in the triangle. Figure 6b shows a single triangulation edge ordering of the edge that does not create a crossing. For the next step, as there is a triangle connecting nodes **∅, bc, abc** with two edges ordered, then the edge labelled **abc** can have an order assigned. We can read around the two edges with label order already assigned, starting from the ∅ node to get a word **bca**. If we order the unassigned edge similarly from the ∅ node, we would get no crossings. Swapping any pair of labels in this edge order would give one contour crossing, for example ordering the contours **b, a, c** along the edge from ∅ to **abc** would mean contours **a** and **c** cross in this triangle. However, in Figure 5a, it can be seen that the order has been defined as **a, b, c** so that **a** crosses with both **b** and **c**.

We take a greedy approach to assigning label ordering to triangulation edges. That is, the first time we encounter a triangle where it is possible to order edge labels in such a way that it will result in contours crossing where that crossing is deemed necessary from the associated dual face, we choose such an ordering. If contour crossings that may introduce extra zones are possible in the triangle (for instance, because two separate pairs of contours may cross in the triangle), then we find the contour label that makes the maximum number of crossings in that triangle. We then ensure that our label ordering means that all crossings in that triangle involve that contour.

It is relatively easy to increase the number of triangles if the granularity of the layout is desired to be finer, although there are other similar mechanisms to improve layout by adding points to polygons independent of triangulation [7]. Figures 7a and 7b illustrate how new triangles might be added, although this is not currently implemented in our system. The edge labelling on the new triangle edges is given by the intersection of labels on the two original triangle edges that the new triangle edge splits.

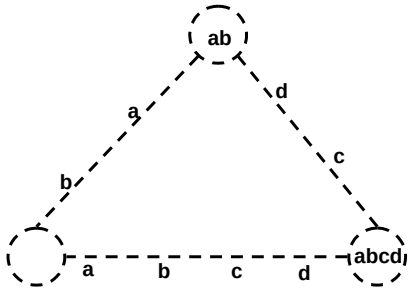

Figure 7a. A triangle face with assigned edges.

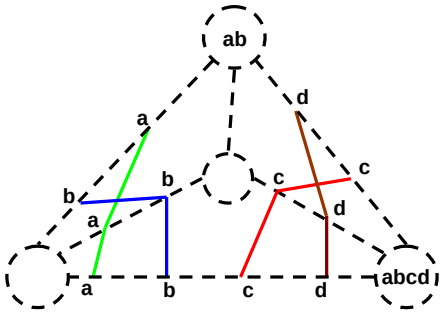

Figure 7b. Routing with extra triangles.

2.4. Non-atomic diagrams

Up to this point this paper has only included examples of atomic diagrams, that is, diagrams where no set of contours are completely disconnected from another set in the diagram. Non-atomic, or nested, diagrams [6] have disconnected components. The above method can be used to draw both atomic and non-atomic diagrams. However, for reasons of algorithmic efficiency, as well as improved layout, it is desirable to lay these diagrams out as separate components, which are joined at a later date. Figure 8 shows a nested diagram: **∅ a b ab ac ad ae acd.**

Nested components can be identified from the dual graph by articulation points in the graph and placed in a maximal rectangle that can found in the appropriate zone, as shown in Figure 8. Where multiple nested components are present in a single zone, the rectangle can simply be split into the required number of sub rectangles. Any nested component may have further nesting present, in which case the process is simply repeated.

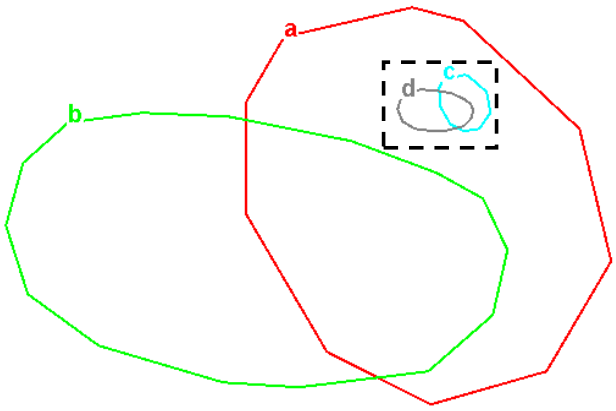

Figure 8. A nested diagram, showing the rectangle (in which the nested components can appear.

3. Layout improvements

The layout of Euler diagrams with the method described in Section 2 leads to diagrams that can have undesirable features, such as line segments that are too close together, or that have jagged edges. We attempt to improve the layout of the diagrams with optimization mechanisms on the triangulated dual. First, the layout of the triangulated graph is improved, and secondly, the positions of the edge labels, where the contours pass through, are modified, whilst maintaining the label ordering assigned to the triangulation edges. We also remove the empty node from the dual to enable the contours to be more evenly distributed around the diagram.

Planar layout mechanisms often lead to poorly laid out graphs, with long, narrow faces. A triangulation obtained from such layouts often has narrow triangles, see Figure 9a. To improve the layout of the triangulated graph we apply a force directed graph drawing algorithm. The result of applying this layout is shown in Figure 9b. This is a spring embedder with an additional force, a node-edge repulsion. The goal is to maintain planarity and to keep the current faces in the graph, whilst improving the regularity of the triangles. This force also evens out the bordering nodes (hidden in Figure 9b) so that they are at a more even distance from the dual graph.

The points where a contour cuts a triangulated edge in both figures 9a and 9b are the places where the labels are positioned; initially, the labels are evenly distributed. However we can attempt to improve the layout of the diagram by moving the edge labels, as long as we do not change their order (which may cause incorrect contour crossings to occur) or bring them too close to each other or the nodes at either end of the edge (possibly making a zone too small to be easily seen). We use a simple heuristic, that of attempting to make each angle on the contour equal, which will tend to make the polygons more regular. This is not an exact heuristic, as different length lines between edge labels can adversely affect the regularity of the polygon. In addition, label positions that indicate a convex contour are heavily penalized, as concave contours tend to make diagrams less understandable. More advanced heuristics for contour layout are given in [7].

The heuristic is applied to each contour point of each contour in turn. Each contour point is moved up and down the triangulation edge to see if an improvement can be found. For each contour, the algorithm traverses the contour in one direction, then reverses the direction in an attempt to avoid large bias. A number of iterations occur and on each iteration an attempt is made to improve the layout of all of the contours in the diagram. On each iteration the distance the contour points are moved is reduced in a cooling schedule. See Figure 9c for an example showing the result of this optimization.

The outer node (node with no label) is essential for checking the connectivity and face conditions and when finding a plane layout, as it must appear in the outer face of the planar graph. However, this node is not required at the contour layout stage, since no contours will enclose it. Once the planar layout is completed, the outer node can be removed and the force directed and edge label redistribution can be applied. If the outer node is present, nodes connect to it with a straight line, so producing an undesired orientation to the diagram. Removing the outside node allows the remaining nodes to be evenly distributed around the graph by the force directed layout method. An example of output when the outer node is removed and the subsequent layout improvements are applied is shown in Figure 9d.

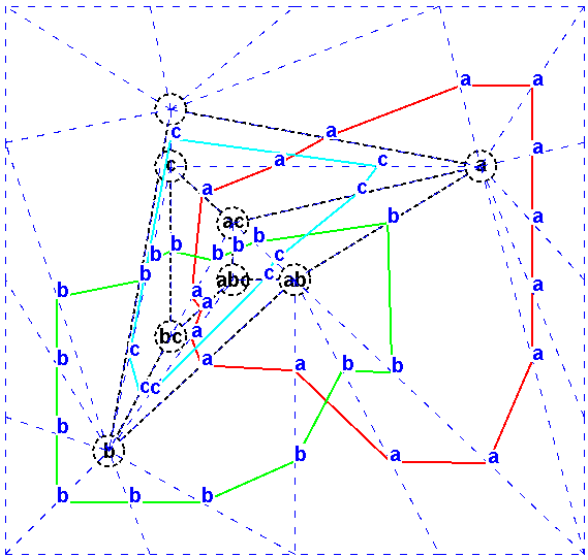

Figure 9a. Triangulated graph after planar layout.

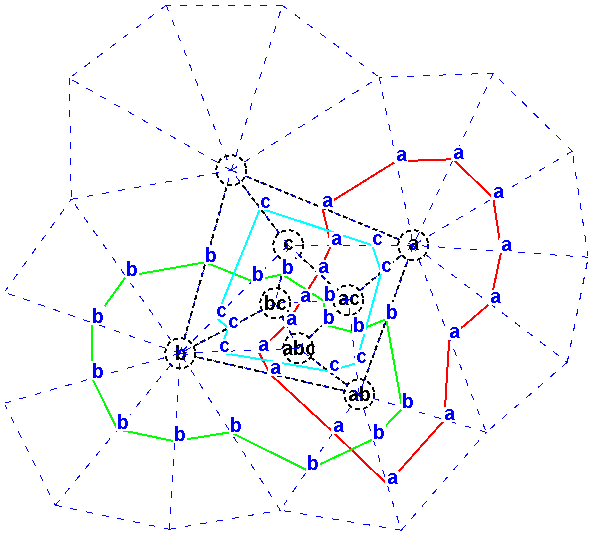

Figure 9b. Triangulated graph after force directed layout.

For any contour with label *l*, we can define the allowed region in which it can be routed. This is formed from the two polygons defined as follows. An inner-polygon (resp. outer-polygon) is formed by joining the edge labels inside (resp. outside) the contour which are on the edges through which the contour passes and immediate neighbours of *l*. The allowed region where contour a can be placed is shown in Figure 10. This region should allow us to form more sophisticated optimizations for the layout in future work.

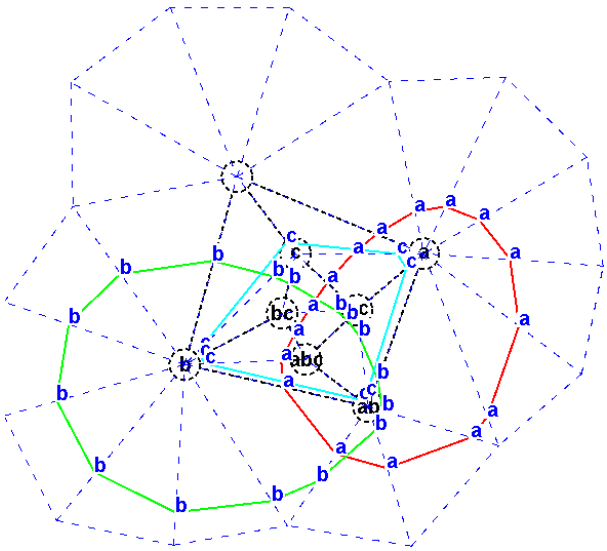

Figure 9c. Triangulated graph after contour point optimization.

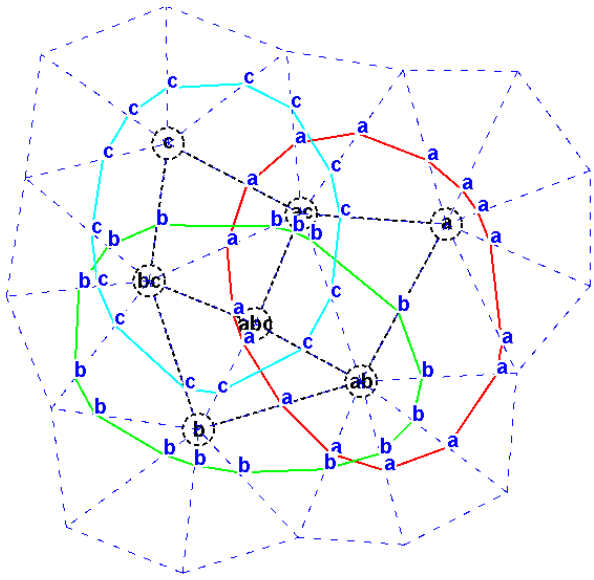

Figure 9d. Removal of outer zone node.

4. Results

In this section we give some examples of output from our software. Figure 12 shows how large diagrams can draw nicely, especially when there are relatively few curve intersections. This diagram, like all in this section, has been drawn with the empty set present.

Figure 13 shows an atomic Euler diagram with six contours. The layout is quite effective despite the relatively complex nature of the diagram. The removal of the empty set has allowed the contours adjacent to the outside zone to be evenly distributed around the diagram.

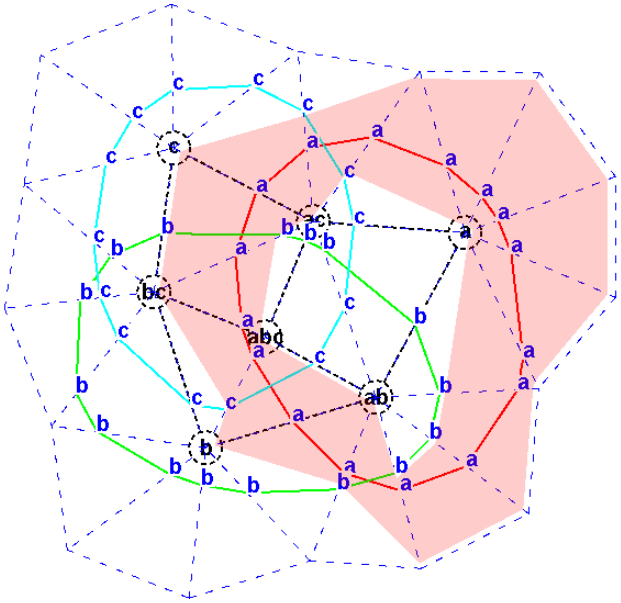

Figure 10. Allowed region for contour a**.**

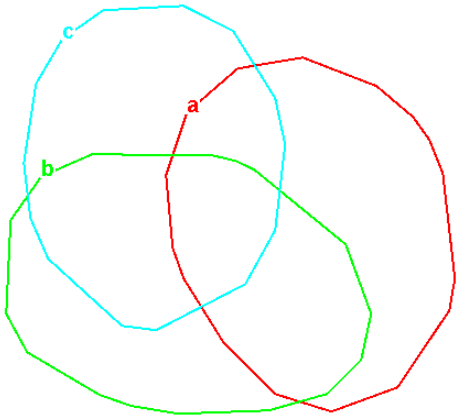

Figure 11. Final diagram for ∅ a b c ab ac bc abc**.**

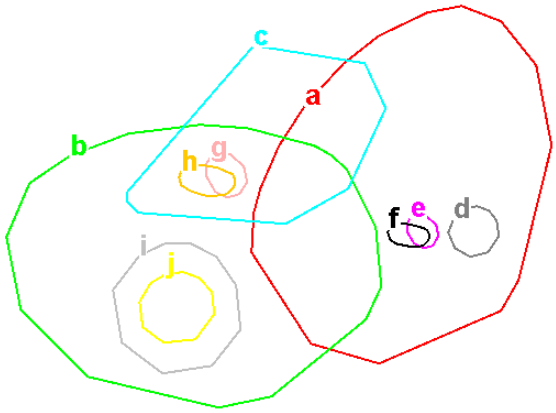

Figure 12. A Euler diagram with ten contours and a number of nested components

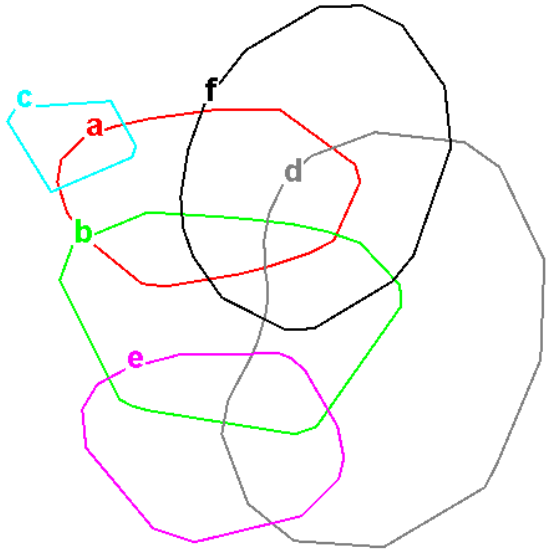

Figure 13. An Euler diagram with six contours.

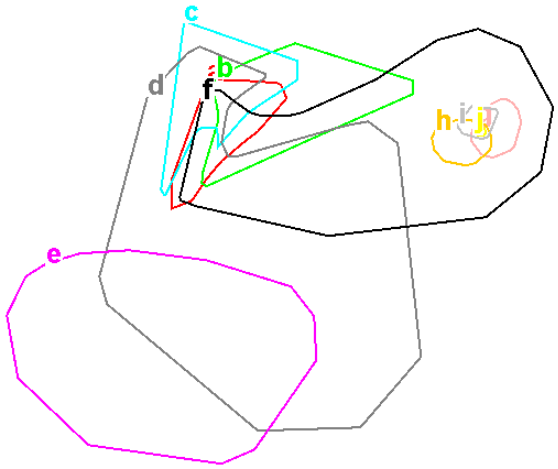

Figure 14. An Euler diagram with ten contours in need of further improvement.

Figure 14 shows a complex Euler diagram. The large number of contours and zones in this diagram means that a simple heuristic to optimise contours in turn begins to prove ineffective. Whilst many of the contours have a reasonable layout, contours **c**, **d** and **f**, which all cross a number of other contours are badly laid out. A compromise layout that makes a great improvement for one contour in exchange for a slightly worse layout for another contour might well improve the overall layout. In addition, there is a contour **j**, in the intersection of contours **g**, **h** and **i**. As the zone it appears in is small, there is no room for **i** to be drawn effectively This illustrates the limitations of not considering the size of zones in which nested diagrams will appear when drawing the parent diagram.

5. Conclusions and future work

We have presented a method for embedding an Euler diagram from a wellformed dual. As far as we know this is the first Euler diagram embedding technique implemented for all such diagrams. We also have presented novel layout improvement techniques that integrates with our embedding method. Current efforts to improve this work include generalizing to diagrams that are not wellformed, as well as further layout improvements.

We consider that this method will be readily applicable to diagrams that are not wellformed. This is not only an embedding problem, but also requires an extension of the generation method to abstract descriptions that might not have dual graphs that pass the conditions described in Section 2.1. Multiple contour crossing points such as triple points can be dealt with by choosing triangles in which to place the multiple points. However, some method needs to be implemented that distinguishes between various multiple points as the number of intersections between contours at a point goes beyond three.

Concurrent contours, where segments of curves run in parallel, can also be present in Euler diagram. A dual that results in a Euler diagram with concurrent contours has edges labelled with more than one contour. Routing two curve segments along the same line is not likely to be problematic. If we wish to generate Euler diagrams where the zones may consist of more than one component of the plane then an approach is to allow dual graphs to have a node for each required zone component.

The choice of triangle in which crossings are placed can be investigated from a usability perspective -- perhaps choosing triangles that make routing the crossing contours most regular would be beneficial; at the moment the choice does not take into account any layout considerations.

We also intend to continue to improve the layout of diagrams from a usability perspective. A simple approach would be to smooth out the current polygons using Bezier curves or similar approximations. In addition, we can already find allowed regions for contours to be drawn within, so that we can attempt to fit shapes such as circles and ovals into the region, rather than polygons. The current layout methods could be improved by the application of previous contour layout work [7] which contains more sophisticated heuristics. Moreover, applying heuristics to movements of groups of edge labels, rather than concentrate on one edge label at a time is likely to bring significant benefits.

6. ACKNOWLEDGMENTS

This work has been funded by the UK Research Council, the EPSRC, under grants EP/E010393/1 and EP/E011160/1.

REFERENCES

[1] S. Chow and F. Ruskey. Drawing area-proportional Venn and Euler diagrams. In GD 2003. LNCS 2912, pages 466–477. Springer, 2003.

[2] R. DeChiara, U. Erra, and V. Scarano. VennFS: A Venn diagram file manager. In Proc. IV03, pages 120–126. IEEE Computer Society, 2003.

[3] A. Fish, J. Flower, and J. Howse. The semantics of augmented constraint diagrams. Journal of Visual Languages and Computing, 16:541–573, 2005.

[4] J. Flower and J. Howse. Generating Euler Diagrams, Proc. Diagrams 2002, LNAI 2317, Springer, 2002.

[5] J. Flower, A. Fish and J. Howse. Jean Flower. Euler Diagram Generation. To appear in the Journal of Visual Languages and Computing, Elsevier, 2008.

[6] J. Flower, J. Howse, and J Taylor. Nesting in Euler diagrams: syntax, semantics and construction, Journal of Software and Systems Modeling, pp 55-67, 2003.

[7] J. Flower, P. Rodgers and P. Mutton. Layout Metrics for Euler Diagrams. Proc. IV03. pp. 272-280. 2003.

[8] P. Hayes, T. Eskridge, R. Saavedra, T. Reichherzer, M. Mehrotra, and D. Bobrovnikoff. Collaborative knowledge capture in ontologies. In Proc. Knowledge Capture, pp. 99–106, 2005.

[9] J. Howse, G. Stapleton, and J. Taylor. Spider diagrams. LMS J. Computation and Mathematics, 8:145–194, 2005.

[10] J. Howse and S. Schuman. Precise visual modelling. J. Software and Systems Modeling, 4:310–325, 2005.

[11] H. A. Kestler, A. Müller, T.M. Gress and M. Buchholz. Generalized Venn diagrams: a new method of visualizing complex genetic set relations. Bioinformatics 21(8) 2005

[12] F. Ruskey. A Survey of Venn Diagrams. The Electronic Journal of Combinatorics. March 2001.

[13] H. Sawamura and K. Kiyozuka. JVenn: A visual reasoning system with diagrams and sentences. In Proc. Diagrams 2000, LNAI 1889. pages 271–285. Springer-Verlag, 2000.

[14] S.-J. Shin. The Logical Status of Diagrams. CUP, 1994.

[15] G. Stapleton, J. Howse, and J. Taylor. A decidable constraint diagram reasoning system. Journal of Logic and Computation, 15(6):975–1008, December 2005.

[16] Tavel, P. Modeling and Simulation Design. AK Peters Ltd. 2007

[17] A. Verroust and M.-L. Viaud. Ensuring the drawability of Euler diagrams for up to eight sets. Proc. Diagrams 2004, Cambridge, UK. LNAI 2980, pp. 128–141. Springer, 2004.