# General Euler Diagram Generation

Peter Rodgers[1], Leishi Zhang[1] and Andrew Fish[2]
[1] Computing Laboratory, University of Kent, UK
[2] Computing Mathematical & Information Sciences, Brighton University of Brighton, UK
P.J.Rodgers@kent.ac.uk, L.Zhang@kent.ac.uk, Andrew.Fish@brighton.ac.uk

**Abstract.** Euler diagrams are a natural method of representing set-theoretic data and have been employed in diverse areas such as visualizing statistical data, as a basis for diagrammatic logics and for displaying the results of database search queries. For effective use of Euler diagrams in practical computer based applications, the generation of a diagram as a set of curves from an abstract description is necessary. Various practical methods for Euler diagram generation have been proposed, but in all of these methods the diagrams that can be produced are only for a restricted subset of all possible abstract descriptions.
We describe a method for Euler diagram generation, demonstrated by implemented software, and illustrate the advances in methodology via the production of diagrams which were difficult or impossible to draw using previous approaches. To allow the generation of all abstract descriptions we may be required to have some properties of the final diagram that are not considered nice. In particular we permit more than two curves to pass though a single point, permit some curve segments to be drawn concurrently, and permit duplication of curve labels. However, our method attempts to minimize these bad properties according to a chosen prioritization.

**Keywords:** Euler Diagrams, Venn Diagrams

## 1 Introduction

Euler diagrams are sets of (possibly interlinking) labelled closed curves and are popular and intuitive notation for representing information about sets and their relationships. They generalize Venn diagrams [16], which represent all possible set intersections for a given collection of sets. Euler diagrams allow the omission of some of these set intersections in the diagram, enabling them to make good use of the spatial properties of containment and disjointness of curves. Euler diagrams are said to be effective since the relationships of the curves matches the set theoretic relationships of containment and disjointness [10]; they provide 'free rides' [17] where one obtains deductions with little cognitive cost due to the representation. For example, if A is contained in B which is contained in C then we get the information that A is contained in C for free.

The motivation for this work comes from the use of Euler diagrams in a wide variety of applications, including the visualization of statistical data [2,13], displaying the results of database queries [1] and representing non-hierarchical computer file systems [3]. They have been used in a visual semantic web editing environment [18] and for viewing clusters which contain concepts from multiple ontologies [11].

Another major application area is that of logical reasoning [12] and such logics are used for formal object oriented specification [14].

A major requirement for many application areas is that they can automatically produce an Euler diagram from an abstract description of the regions that should be present in the diagram. This is called the Euler diagram generation problem.

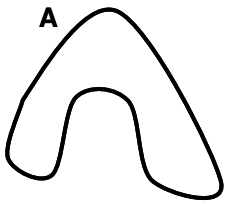


**Fig. 1a.** Concave Curve.

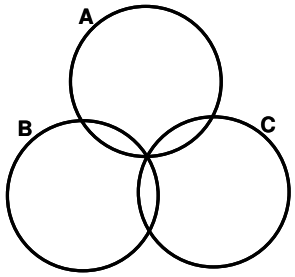


**Fig. 1b.** A triple point.

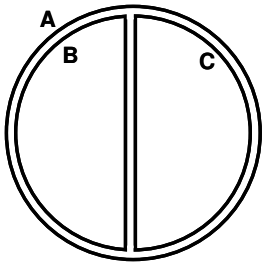


**Fig. 1c.** Concurrent curves.

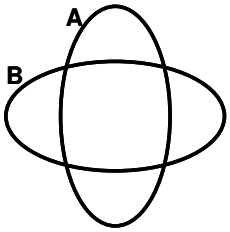


**Fig. 1d.** Disconnected zones.

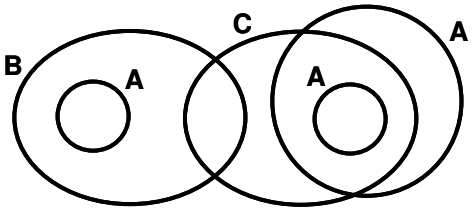


**Fig. 1e.** Duplicated curve label.

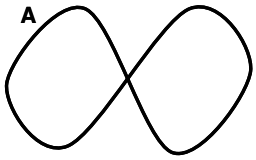


**Fig. 1f.** Non simple curve.

A range of diagram properties, called wellformedness conditions, which are topological or geometric constraints on the diagrams, have been suggested with the idea of helping to reduce human errors of comprehension; these properties can also be used as a classification system. Some of the most common properties are shown in Fig. 1. Note that the term *zone* in Fig. 1d refers to the region enclosed by a particular set of curve labels, and excluded by the rest of the curve labels. For example the region that is inside the curve labelled A but outside the curve labelled B is disconnected here.

The definition of what constitutes an Euler diagram varies in the literature, and can usually be expressed in terms of these wellformedness conditions. Although Euler himself [5] did not formally define the diagrams he was using, his illustrations do not break any of wellformedness conditions given in Figure 1. In [7] the first Euler diagram generation algorithm was presented and further formalized in [6]. This work guaranteed the production of an Euler diagram that meets all of the wellformedness conditions in Fig. 1 except Fig 1a, from an abstract descriptions whenever it was possible to do so. An implementation of the algorithm was also provided which had a limited guarantee of being able to draw any such diagram with up to four contours in any one connected component. In [1] the relaxation of the wellformedness conditions to allow multiple points and concurrent contours was adopted, and although no conversion from theory to practise was provided, it was shown that the Euler diagram generation problem is NP-Complete in this case. Since imposing some wellformedness conditions implies that some abstract descriptions are not realisable as Euler diagrams, in [1] the notion of an Euler diagram was extended so that any abstract description with at most nine sets could be drawn: they used Euler diagrams that had holes, which are a restricted version of allowing duplicate curve labels.

In this paper, we integrate and significantly extend the work of these three major attempts at the Euler diagram generation problem and provide a complete solution to general Euler diagram generation in the sense that any abstract description is drawable using our method.

We define an *Euler diagram* to be a set of labelled closed curves in the plane. We call the set of all of the labelled curves with the same label a *contour*. A *zone* of an Euler diagram is a region of the plane enclosed by a set of contours, and excluded by the rest of the contours. The diagrams obtained via our generation process can have curves of any geometric shape and they may have duplicate contour labels, multiple points, and concurrent curves. However, we guarantee not to generate any diagrams with duplicate zones or non-simple curves.

Utilising this broad definition of Euler diagrams makes the general generation problem of any abstract description possible, but typically, the “more non-wellformed” a diagram is the less desirable it is from a usability perspective. Therefore, we adopt a strategy which guides the output towards being as wellformed as possible, according to a chosen prioritisation of the wellformedness conditions, whilst ensuring that we generate a diagram with the correct set of zones (i.e. it complies with the abstract description). However, we give no guarantee that the diagrams generated are the most wellformed diagrams possible since some of the problems that need to be solved to ensure this are NP-Complete.

In this paper we adopt the convention of using single letters to label contours. Each zone can be described by the contour labels in which the zone is contained, since the excluding set of contour labels can be deduced from this set. An abstract description is a description of precisely which zones are required to be present. For example, the abstract description for the Euler diagram in Figure 2b is **∅ b c ab ac abc**, where **∅** indicates the zone which is contained by no contours, called the *outside zone*, which must be present in every abstract description.

In Section 2, we give details of the generation method. Section 3 gives our conclusions and future directions.

## 2 The Generation Process

First we give a high level outline of the methodology used, with details and explanation of the terminology appearing in later sections. Given an abstract description of an Euler Diagram, we produce an embedded Euler diagram using the following steps:

1. Generate the superdual graph for the abstract description.
2. Using edge removal, find a planar subgraph that is either wellconnected or close to wellconnected.
3. If the graph is not wellconnected, add concurrent edges to increase the closeness of the graph to being wellconnected whilst maintaining planarity.
4. Find a plane layout for the graph.
5. Add edges to reduce unnecessary tangential intersections, forming the dual of the Euler graph.
6. Find subgraphs where duplicate curves will be required.
7. Construct the Euler diagram from the dual of the Euler graph using a triangulation based method.

Since we are constructing the dual of the Euler graph, planarity is clearly essential. If the dual graph constructed is not wellconnected then the Euler diagram will have either duplicate curves or concurrency. Steps 2 and 3 try to reduce the instances of either. However, step 3 may add concurrent edges (i.e. those with multiple contour labels) which can reduce the number of duplicate curves used at the expense of causing concurrency. Step 5 removes unnecessary tangential intersections (those that can be removed without introducing concurrent curve segments). Checking the face conditions, as in [7] would identify if multiple points will appear, but since attempting to search for an Euler dual which passed the face conditions (and so has no multiple points) is so time consuming, we omit this step.

### 2.1 Generating a Super Dual

Recall that an abstract description of a diagram is list of zone descriptions (which are just the sets of contour labels that will contain the zones). As described in [7] we can construct the superdual by taking one node for each required zone, and labelling each one with its zone description. When an Euler diagram is drawn, each contour's curves will enclose the nodes whose label set contains the contour label. Two nodes in the superdual are connected by an edge precisely when the labels differ by a single contour label. The edges are labelled with the difference between their incident node labels. Fig. 2 shows an example of a superdual, and resultant Euler diagram generated for the abstract description **∅ b c ab ac abc**. In this case, and for other small examples that can be drawn without concurrent contours or duplicate curve labels, the superdual can be embedded without requiring steps 2,3,5,6 of our process. However, many superduals are not planar, and so methods to find a planar dual need to be applied, as described in the next section.

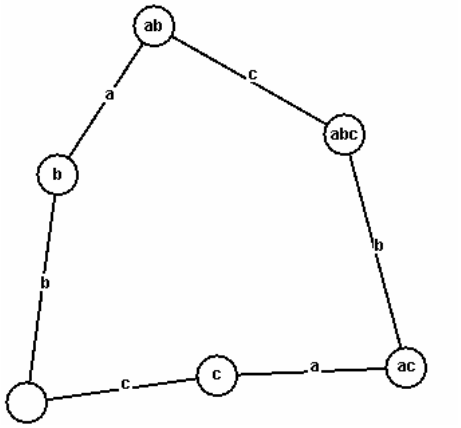


**Fig. 2a.** Superdual for **∅ b c ab ac abc**

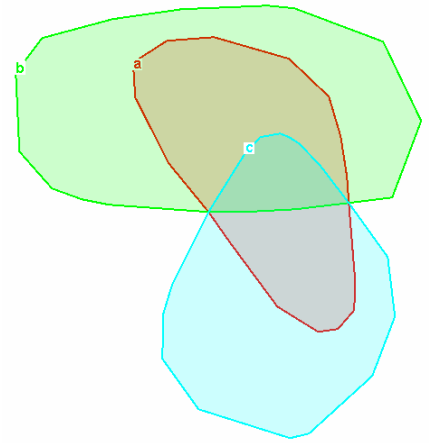


**Fig. 2b.** Embedded diagram.

### 2.2 Edge Removal to Achieve Planarity

Given a superdual that is non-planar, we try to find a planar subgraph of the superdual that can be used to generate a general Euler diagram that has no concurrency or duplicated curve labels; i.e. it must be wellconnected, which means that it must pass the connectivity conditions below. Even if such properties are necessary, the amount of concurrency and the number of curves in a contour may be reducible (by finding a subgraph that is "close" to passing the connectivity conditions). The connectivity conditions state that

*a*. the graph is connected

and for each contour label in the abstract description:

*b*. if the nodes with that contour label present are removed (recall, a node is labelled by a collection of contour labels) then the graph remains connected and,

*c*. if the nodes without that contour label present are removed then the graph must also remain connected.

If condition *a* does not hold in the superdual, then concurrency is required in the Euler diagram, and Step 3 of our method will be applied to add a multiply labelled edge to the dual (corresponding to concurrency of edges in the Euler diagram). If conditions *b* or c do not hold and they cannot be fixed by the addition of edges without breaking planarity, then duplicate curve labels will be used for that contour – in the case of condition c failing, curves are placed "inside" another curve of that contour, forming holes in the contour.

In Step 2 we attempt to find a wellconnected planar dual by removing edges from the superdual. If this cannot be done, our edge removal strategy attempts to find a planar dual that has as much connectivity as possible, that is the occurrences of conditions *b*, or *c* are as few as can be achieved.

To guarantee to find a wellconnected planar dual where one exists is an NP-Complete problem [1]. Therefore we resort to heuristics to do as good a job as

possible. Our current technique is to take a fairly lightweight approach of discovering potentially removable edges, checking those that may be removed from the dual and exploring the effects of removing combinations of these. We first layout the superdual graph using a spring embedder [4] and remove highly crossed edges, preferring the potentially removable edges. Once a planar layout has been found we then attempt to add back any unnecessarily deleted edges that improve the wellconnectedness. This paper does not focus on heuristics, and we give only a simple demonstration of a possible technique. As with other NP-Complete problems we expect there to be a number of alternative heuristics.

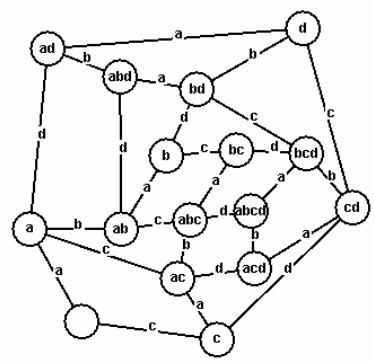


**Fig. 3a.** Planar dual for 4Venn

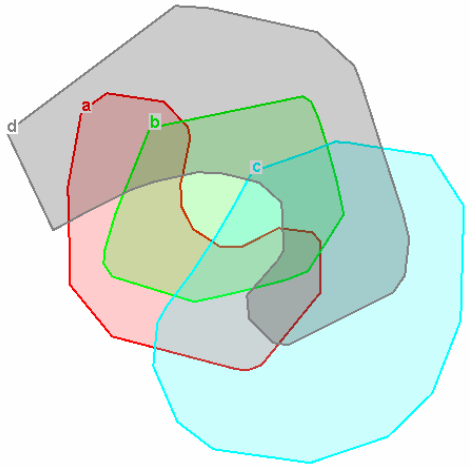


**Fig. 3b.** 4Venn embedded.

Fig. 3a shows a wellconnected plane dual for 4Venn (the Venn diagram on 4 sets), and the corresponding diagram generated from this dual is shown in Fig 3b. Various edges have been removed from the superdual to achieve planarity, including the edge between ∅ and **b**, and the edge between **c** and **bc**. Depending on the starting conditions, and on how much time is given to the search, it is also possible that versions of 4Venn which have triple points and duplicate curve labels might be created (see Fig. 8 for example).

### 2.3 Concurrent Edge Addition

If, after Step 2, the graph is disconnected then, in Step 3, we attempt to make it connected by adding edges whilst maintaining planarity. In addition, for each contour label, if removal of nodes without that contour label present would result in multiple disconnected subgraphs, then we attempt to add edges which would connect those subgraphs.. Similarly, we attempt add edges which connect any multiple disconnected subgraphs formed by the removal of nodes with that contour label present. Recall that edges in the dual graph are labelled with the difference between the labels of the nodes they are incident with. Edges that are labelled by more than one contour label are called *concurrent edges* since they correspond to the use of concurrency in the

Euler diagram. Adding edges in this manner can reduce the number of duplicate curve labels but can also add extra concurrency.

Given that there appears to be a combinatorially explosive number of possible ways of connecting up the various subgraphs of the dual graph, and only one of which might be optimal, we expect that the problem of finding a planar dual which is as close to wellconnected as possible by adding edges to be at least NP-Complete. Hence, we take a heuristic approach to deciding how to add edges. Again we adopt a simple method, taking a greedy approach, but with the small examples we are currently exploring (less than 10 sets) we find that relatively few disconnected components appear. We take one disconnected component and attempt to connect it to another by an edge that is labelled with the least number of contours. This process continues until the dual is connected or no more improvements can be made.

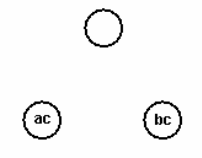


**Fig. 4a.**
Superdual of **∅ bc ac**

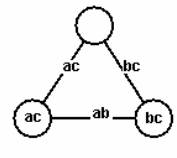


**Fig. 4b.**
With concurrent edges

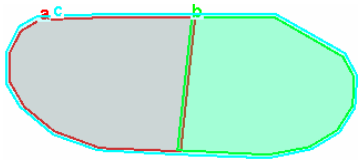


**Fig. 4c.**
**E**mbedded diagram.

An example of the process is shown in Fig 4 for the abstract description **∅ ac bc**. Fig 4a shows the superdual which is disconnected; no two nodes have label sets differing by one label. Fig 4b shows the dual graph with concurrent edges added as a result of Step 3. We note that adding any two edges to the superdual does not make the graph wellconnected. For example, if we only added one edge between nodes labelled "∅" and "ac"**,** and another edge between nodes labelled "∅" and "bc" then for the contour labelled "c" condition *c* of the connectivity conditions does not hold since the nodes "ac" and "bc" would not be not adjacent. Similarly, leaving either of the pair of nodes labelled "∅" and "ac" or the pair of nodes labelled "∅" and "bc" not adjacent breaks condition *b* of the connectivity conditions. The Euler diagram created (using the dual graph in Fig 4b) is shown in Fig. 4c, where we slightly separate the concurrent curve segments to ease comprehension.

### 2.4 Planar Layout

In this step we embed the dual graph in the plane. There are various standard approaches to planar layout. At the moment we use a method provided by the ODGF software library. We make one adjustment to ensure the node labelled with "∅" is in the outer face of the drawn graph, as this node represents the part of the diagram enclosed by no contour. The layout of the dual has a significant impact on the drawing of the diagram, and Section 3 includes some discussion of methods to layout planar graphs to improve the usability of the final diagram.

## 2.5 Edge Addition to Remove Tangential Intersections

For the purposes of embedding we treat the dual as the dual of an Euler graph [1]. An Euler graph can be formed from an Euler diagram by placing a node at each point where curves meet or cross, and connecting them up with edges that parallel the curve segments. Using the dual of an Euler graph means that, unlike the treatment of the dual in [7], each face in the dual has at most one point where contours meet or concurrent edges separate. However, it can lead to the introduction of unnecessary tangential intersections and so we apply an edge addition process to remove them (subdividing the faces of the dual separates the tangential meetings of the curves).

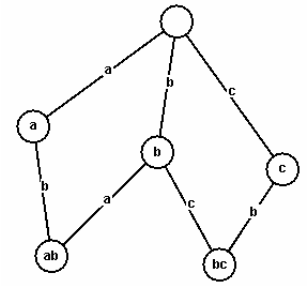


**Fig. 5a.** Dual graph for ∅ **a ab b bc c**

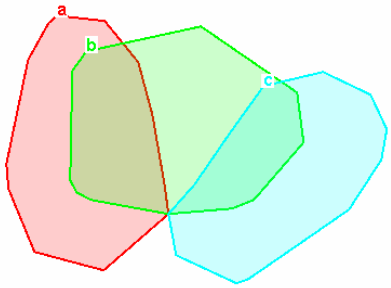


**Fig. 5b.** Without edge addition.

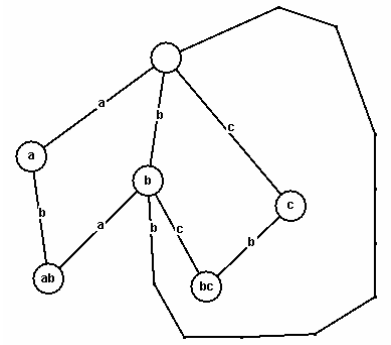


**Fig. 5c.** Additional edge between ∅ and **b**

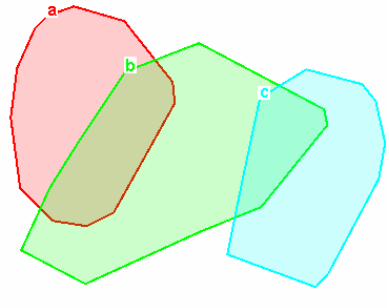


**Fig. 5d.** With edge addition.

We detect the need for extra edges by testing each face in the dual graph. If it is possible to add an edge between two non adjacent nodes in the face and the new edge will be labelled with one of the edge labels of the face, then that edge is added (recall that edges are labelled with the difference in their incident node labels). An example is shown in Fig. 5, where Fig. 5a shows a dual graph and Fig. 5b shows the corresponding Euler diagram which contains an unnecessary tangential intersection (the point where all of the three curves meet). The graph in Fig. 5a has an outside face that has an edge labelled "b", but it can also have another edge labelled "b" added to it between nodes labelled ∅ and **b**, as shown in Fig. 5c. The Euler diagram constructed from the dual with the additional edge is shown in Fig. 5d. We route this edge without bends if possible, but often it is not possible, as is the case in Fig. 5c. In this case, a triangulation of the face is made, and the edge is routed through appropriate triangulated edges.

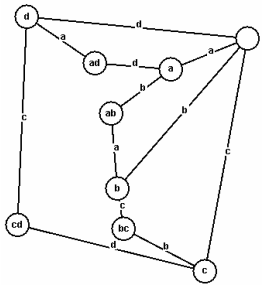


**Fig. 6a.** Dual of the Euler graph for **∅ a b c d ab ad bc cd**.

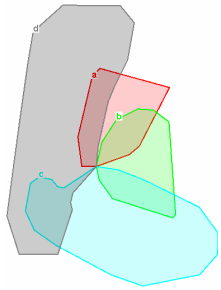

**Fig. 6b.** Embedded diagram.

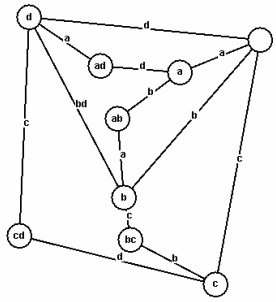


**Fig. 6c.** Additional edge between **b** and **d**

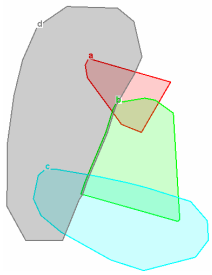

**Fig. 6d.** Diagram with reduced tangentiality but extra concurrency.

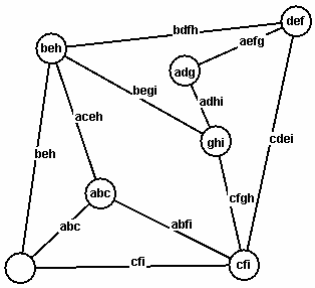


**Fig. 7a.** Dual of the Euler graph for **∅ abc def ghi adg beh cfi**

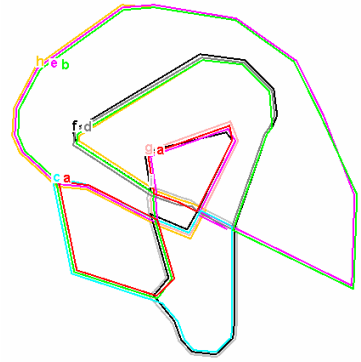

**Fig. 7b.** Embedded diagram.

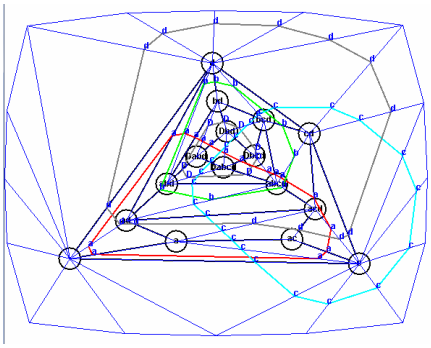

**Fig. 8a.** Contour Routing

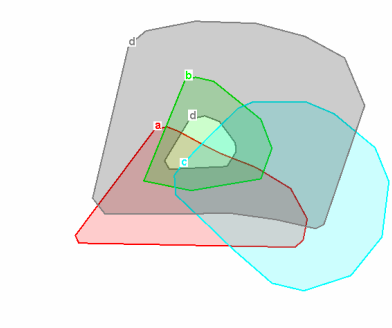

**Fig. 8b.** Embedded diagram.

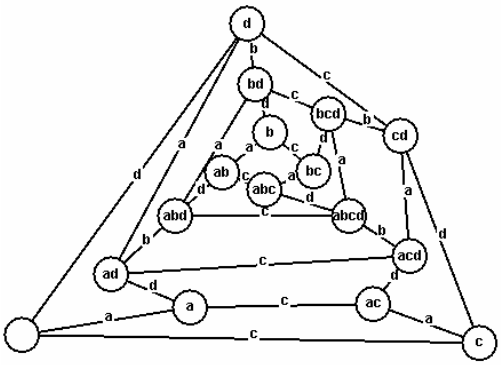


**Fig. 8c.** Plane dual for 4Venn

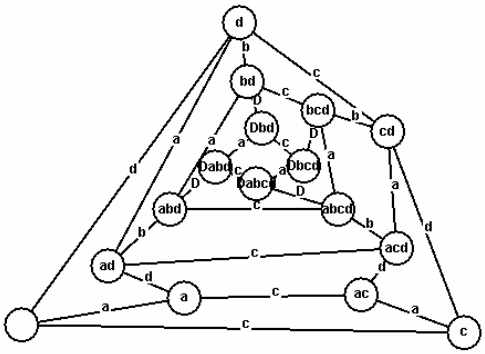


**Fig. 8d.** Hole has labels "D" and "d" added.

Not all tangential intersections are removed by this method because some can only be removed at a cost of adding extra concurrency. For example Fig. 6a shows the dual of the Euler diagram in Fig 6b. Addition of an edge labelled "bd" between nodes labelled "b" and "d" would result in the removal of the tangential connection between the nodes labelled "a" and "c", however this would also result in a concurrent segment "bd" shown in figures 6c and 6d.

### 2.6 Duplicate Curve Labels

As shown in [1], not all Euler diagrams can be embedded in the plane with simple, uniquely labelled curves. The reason is that there are abstract descriptions for which any wellconnected dual graph is non-planar. The example in the paper is **∅ abc def ghi adg beh cfi**, for which any wellconnected graph with the corresponding nodes contains a subgraph that is isomorphic with the non-planar $K_{3,3}$. This limits the method of [1] to guarantee the existence of a drawing only if there are 8 sets or less. We demonstrate that this example can be drawn by our method in Fig. 7b; the red contour "a" has two curves, since the subgraph of the dual with nodes containing the label a consists of the two nodes **abc** and **adg** which not being adjacent in Fig. 7a, breaking condition *c* of the connectivity conditions. Given the need for large amounts of concurrency when drawing this diagram, it is not likely to have a particularly usable embedding, but this example demonstrates the ability of our method to embed a diagram from any abstract description.

The example in Fig.7 uses duplicate curves for the same contour. Given a dual graph (obtained from Step 3 of our method), we discover the duplicate contours required by looking at the connected components of the subgraphs of the dual that include the contour label present (corresponding to the wellconnected condition *c*, Section 2.2) or removed, corresponding to the use of holes (wellconnected condition *b*). To enable us to draw the Euler diagram, we re-label the nodes of the graph that contain the contour label which requires the use of duplicate curves, being careful to distinguish the case of holes. Essentially, we keep the label of the contour the same for one of the components (in the label present case) and change it for the other components (thereby assigning new curve labels; here we adopt the convention of

using capital letters for the duplicates to help distinguish from the usual lowercase). Then we alter the labels of the nodes on the components in the label removed case, so that they indicate the new curve labels assigned as well as the fact that these nodes are within a hole in that contour. This relabeling procedure allows us to draw the curves correctly, but when labelling the contours of the final diagram, we revert to the original labels for the curves.

Fig. 8 shows an example for 4Venn drawn with a hole. Here there is a duplicate label “d” required for two curves, because, when nodes including “d” are removed from the dual graph in Fig 8c the subgraph with nodes labelled “b” “ab” “bc” and “abc” is not connected to the rest of the graph. Therefore, these four nodes have the label “d” added, together with the label “D”, indicating a hole, as shown in Fig 8d. “D” will be mapped back to “d” when the diagram is embedded, as in Fig 8b.

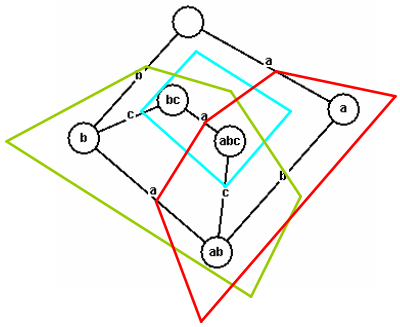


Fig. 9. **An incorrect embedding for** ∅ a b ab ac abc**.**

### 2.7 Constructing the Euler Diagram from the dual of the Euler graph

In general, straight lines cannot simply be drawn between edges of the dual to indicate where the contours pass through the faces, because a face may not be convex. This could cause the lines to intersect edges of the dual graph whose labels do not include the same contour label, possibly introducing incorrect contour intersections that cause the diagram generated to not have the required zone set. If an arbitrary polyline routing through the face is taken, incorrect intersections can again occur, also possibly failing to form a diagram with the required zone set. For example, see Figure 9 where the zone **c** appears but does not exist in the abstract description, and the zone **a** is disconnected, appearing both at the bottom and top right of Figure 9. 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.

First, we triangulate the bounded faces of the plane dual graph, and for the outer face we form a border of nodes with empty labels around the graph and triangulate the polygon that is formed (see Fig. 10d, where the border nodes have been hidden). As with the dual graph, each triangulation edge is labelled with the difference between

the labels present in its incident nodes, see Figure 10a. Again, as with the dual, the labels on the triangulation edges indicate which contours will cross them when we produce an embedding.

We choose one triangle in each face to be the *meeting triangle* in which all contours in that face will cross or meet. In the current implementation, this is taken to be the triangle that contains the centriod of the polygon formed from the face (or is the triangle closest to the centriod, if none contain it). We mark a point called the *meeting point* in the centre of the meeting triangle, and all contours in the face must pass through that point.

Next we assign an ordering of the contours which must pass through each triangulated edge in the face. This will enable us to assign points on the triangulation edges where the contours cross them. For the purposes of this method we add triangulation edges to parallel dual graph edges. Concurrent contours that are drawn across the face maintain concurrency until at least the meeting point, where they may separate if that concurrency is not maintained in the face.

The ordering of contours that pass through a triangulation edge that parallels a dual edge is trivial because there is either only one contour or group of concurrent contours. Also, any triangulation edge with no contours passing through it can be trivially assigned an empty order. It is then necessary to assign a contour ordering to the other triangulation edges of the face. Fig. 10 shows an example of this process. Once the trivial above triangulation edge orderings have been performed there will be at least two triangles with two triangulation edges assigned, see Fig. 10a. If the face is not a meeting triangle (shown as the triangle containing a green circle as the marked point) then the order of the third triangulation edge of such triangle can be assigned; Fig. 10b shows the assignment of one of these. This third triangulation edge will have contours ordered to avoid any contour crossings in the face by reading the order of the two assigned triangulation edges in sequence and using a similar order for the edge, as shown in Fig. 10c where both triangulation edges now have an assigned order. In addition, we enforce the condition that all contours on the other two triangulation edges must also be present in the third triangulation edge, to ensure that all contours reach the meeting point.

The assignment of a contour ordering on a triangulation edge means that another triangle has an additional triangulation edge with contour ordering assigned, as each triangulation edge (that is not a dual graph edge) is shared between two triangles in the face. Hence the process continues until all triangulation edges are assigned an order. At this point the meeting triangle should also have all three of its triangulation edges with assigned order, as the triangles that surround it should all have triangulation edges with assigned order.

This method can be shown to terminate due to the restricted nature of the triangulation, where any triangles without any triangulation edges parallel to face edges imply that there is an extra triangulation face with two triangulation edges parallel to face edges.Once the triangulation edges have been assigned an ordering, the curves can be routed around the face by linking up the appropriate triangulation edge points, except where the triangulation face is the meeting triangle, where they must first pass through the meeting point in the triangle (shown as a filled circle inside a triangle in figures 10a, 10b and 10c).

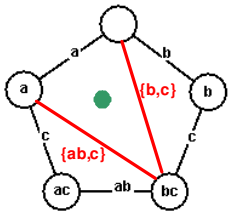


**Fig. 10a.** Unassigned Triangulation Edges

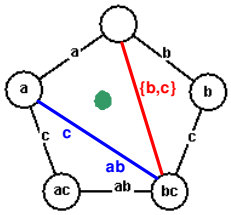


**Fig. 10b.** One Triangulation Edge Assigned.

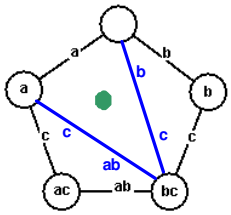


**Fig. 10c.** Both Triangulation Edges Assigned

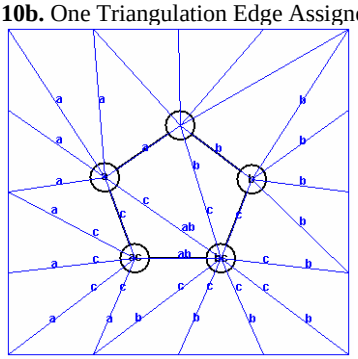

**Fig. 10d.** Every Triangulation Edge

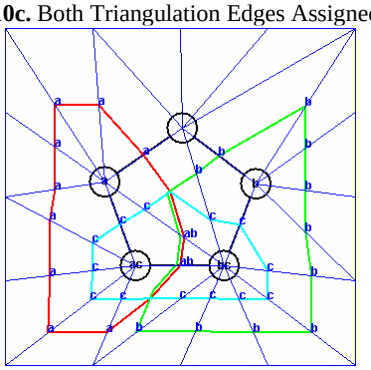

**Fig. 10e.** Contours Routed Through Cut Points

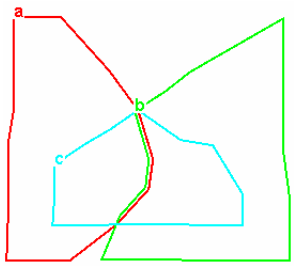


**Fig. 10f.** Final Diagram

### 2.8 Non-Atomic Diagrams

Up to this point we have only shown examples of atomic diagrams, which are diagrams that can be drawn with disconnected contours. [8].. The above method can be used to draw both atomic and non-atomic diagrams, with the atomic components tangentially connected. 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 11 shows a non-atomic diagram that has nested components: **∅ a b ab ac ad ae acd.**

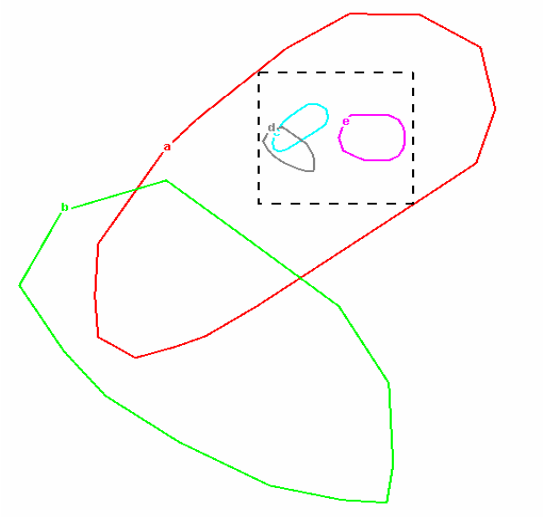


**Fig. 11.** A nested diagram, showing the rectangle in which the nested components can appear.

Non-atomic components can be identified from the dual graph from the abstract description [1] and placed in a maximal rectangle that can found in the appropriate zone, as shown in Figure 11. Nested components can also be created when there is more than one curve in a contour and the additional curves do not intersect with any other curves in the diagram. Where multiple nested components are to be embedded within a single zone, the rectangle is simply split into the required number of sub rectangles. Any nested component may have further nested components inserted by simply repeating the process.

## 3 Conclusions and Further Work

We have presented the first generation method for generating an Euler diagram for any abstract description.. To do this we have brought together and extended various approaches in the literature, and developed new mechanisms for the embedding process. We have demonstrated these ideas with output from a working software system the implements the method. In terms of the methodology adopted, Step 2 - edge removal to find a plane dual, and Step 3 - adding concurrent edges, are computationally intractable problems to solve exactly in the general case, so improved heuristics and optimizations are a rich area of further work. Initially, utilizing effective search techniques such as constraint satisfaction and adapting well known heuristics such as insertion methods from the Travelling Salesman Problem are likely to improve current performance.

A further avenue of research is in improvements of the final layout which is an essential feature in usability terms. Methods, such as those discussed in [9,15] have been applied to the some of the diagrams shown in this paper, and further heuristics that more accurately measure contour smoothness, and as well as measuring other

aesthetic features of the diagram not currently considered could be introduced. Also, the plane embedding of the dual has significant impact on the usability of the drawing, and methods to control the layout at Step 4 could impact on the number of triple points generated, which is currently not restricted, for example.

**Acknowledgments.** This work has been funded by the EPSRC under grant refs EP/E010393/1 and EP/E011160/1.

## References

1. S. Chow. Generating and Drawing Area-Proportional Venn and Euler Diagrams. PhD Thesis. University of Victoria, Canada. 2008.
2. S. Chow and F. Ruskey. Drawing area-proportional Venn and Euler diagrams. In Proc. of Graph Drawing 2003. LNCS 2912, pages 466–477. Springer-Verlag, September 2003.
3. R. DeChiara, U. Erra, and V. Scarano. VennFS: A Venn diagram file manager. In Proc. IV03, pages 120–126. IEEE Computer Society, 2003.
4. P. Eades. A Heuristic for Graph Drawing. Congressus Numerantium, 22. pp. 149-160. 1984.
5. L. Euler. Lettres à une Princesse d'Allemagne, vol 2. 1761. Letters No. 102–108.
6. J. Flower, A. Fish, J. Howse. Euler Diagram Generation. Journal of Visual Languages and Computing, Elsevier, 2008.
7. J. Flower and J. Howse. Generating Euler Diagrams, Proc. Diagrams 2002, LNAI 2317, Springer Verlag, 61-75. 2002.
8. 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.
9. J. Flower, P. Rodgers and P. Mutton. Layout Metrics for Euler Diagrams. Proc. IEEE Information Visualization (IV03). pp. 272-280. 2003.
10. C. Gurr. Effective diagrammatic communication: Syntactic, semantic and pragmatic issues. Journal of Visual Languages and Computing 10, 4, 317–342. 1999.
11. P. Hayes, T. Eskridge, R. Saavedra, T. Reichherzer, M. Mehrotra, and D. Bobrovnikoff. Collaborative knowledge capture in ontologies. In Proc. of 3rd International Conference on Knowledge Capture, pp. 99–106, 2005.
12. J. Howse, G. Stapleton, and J. Taylor. Spider diagrams. LMS J. Computation and Mathematics, 8:145–194, 2005.
13. 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.
14. S.-K. Kim and D. Carrington. Visualization of formal specifications. Proc. APSEC '99, pp. 102-109.
15. P.J. Rodgers, L. Zhang, A. Fish. Embedding Wellformed Euler Diagrams. To appear in Proc. Information Visualization (IV08).
16. F. Ruskey. A Survey of Venn Diagrams. The Electronic Journal of Combinatorics. March 2001.
17. A. Shimojima. Inferential and expressive capacities of graphical representations: Survey and some generalizations. In Diagrammatic Representation and Inference: proceedings of Diagrams 2004, LNAI 2980, pp. 18–21, Springer.
18. P. Tavel. Modeling and Simulation Design. AK Peters Ltd. 2007.
19. 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