Dynamic Euler Diagram Drawing

Peter Rodgers
Computing Laboratory
University of Kent, UK
+44 1227 827913
P.J.Rodgers@kent.ac.uk

Paul Mutton
Computing Laboratory
University of Kent, UK
+44 1227 762811
pjm2@kent.ac.uk

Jean Flower
Visual Modelling Group
University of Brighton, UK
+44 1273 642410
J.A.Flower@brighton.ac.uk

ABSTRACT

In this paper we describe a strategy to lay out a graph-enhanced Euler diagram so that it looks similar to an already existing drawn graph-enhanced Euler diagram. This task is non-trivial when the underlying structures of the diagrams differ. In particular, if a structural change is made to an existing drawn diagram, our work enables the presentation of the new diagram with minor disruption to the user's mental map. As the new diagram is often generated from an abstract representation, its initial embedding may not be close to that of the original, so we have developed similarity measures for Euler diagrams integrated into a multi-criteria optimiser, and a force model for associated graphs that attempts to maintain the original layout. This work extends the two stage non-dynamic graph-enhanced Euler diagram drawing method developed by the investigators.

We apply the dynamic drawing method to diagrammatic reasoning proof sequences, an application area where new diagrams can be generated automatically, without any known embedding. However, dynamic layout methods can be applied to these new diagrams because they are modifications of an original diagram presented by the user.

Keywords

Euler Diagram; Graph Drawing; Dynamic Layout; Hypergraph.

1. INTRODUCTION

Euler diagrams generalise Venn diagrams, having *contours* drawn as simple closed curves which can intersect, contain or exclude other contours. The parts of the plane distinguished by being contained in some contours but excluded from all others are called *zones*. In Figure 2 there are two Venn diagrams shown and in Figure 3 we have a Venn diagram and an Euler diagram.

A *graph-enhanced Euler diagram* comprises an underlying Euler diagram with a graph super-imposed. The nodes of the graph are associated with Euler diagram zones. Two different drawings of the same graph-enhanced Euler diagram are shown in Figure 1.

Graph enhanced Euler diagrams combine the immediate associative readability of graphs with the powerful grouping and set intersection features of Euler diagrams. Many diagrammatic applications have extended graph syntax such as higraphs or hypergraphs, where nodes are grouped in intersecting regions. These structures can be represented as enhanced Euler diagrams and in most of the applications the graph and grouping is liable to change, because of changes to the underlying data structure, or because a user has edited the diagram.

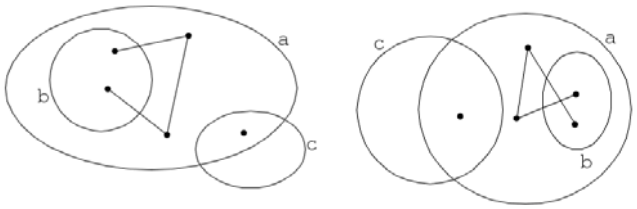


Figure 1. Two drawings of a graph-enhanced Euler diagram

The dynamic diagram application presented in this paper is that of laying out a sequence of diagrams that represents a diagram proof, where the reasoning steps give structural changes in the diagram. To maximise the readability of the proof, consecutive diagrams should appear similar apart from changes made by the reasoning steps.

Dynamic diagram drawing deals with the automatic layout of diagrams when the underlying structure of an *original* diagram has changed to give a *new* diagram. Because of the wide variety of changes possible and reasons for the changes, the task of dynamic drawing is demanding. Techniques for dynamic graph drawing have been explored (see Section 2 for a summary), however no previous work has been performed in dynamic Euler diagram drawing, or dynamic drawing of Euler diagrams enhanced with graphs.

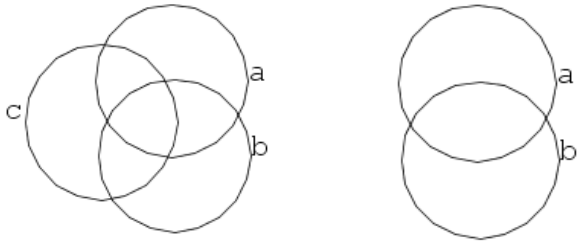


Figure 2. Contour addition / removal

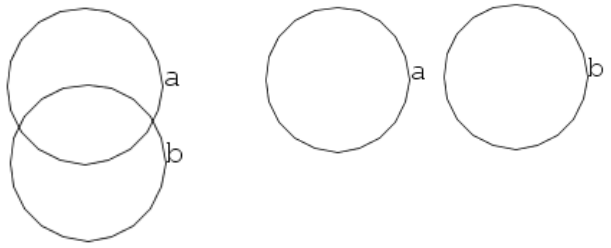
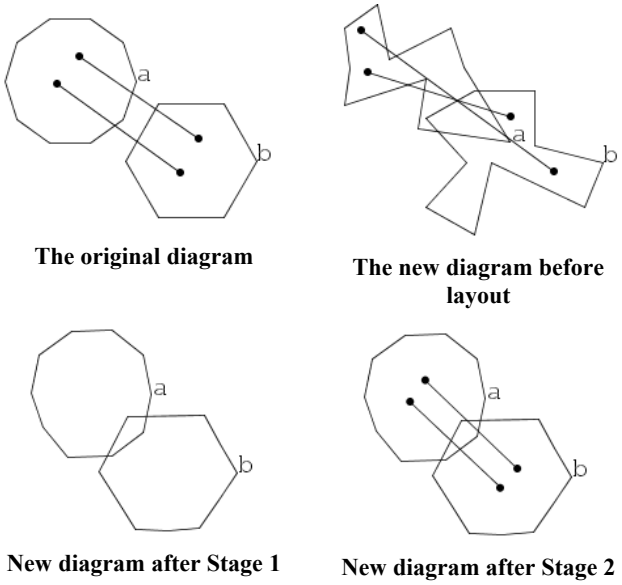


Figure 3. Zone ab is present in only one diagram

Possible changes to an Euler diagram include the addition or removal of contours (see Figure 2), or changes to the zone set (see Figure 3).

In addition to changes to the underlying Euler diagram, the graph associated with the diagram may change with node or edges being deleted or added. Often, both Euler diagram and graph change at the same time.

We have implemented a two stage system for dynamic drawing of Euler diagrams enhanced with graphs that builds on the static drawing method described in [15], however the method described in this paper can be applied to any initial layout, either hand drawn or drawn with a static method. Examples in Section 4 show both of these cases. Firstly we lay out the underlying Euler structure in the new diagram, using information about the layout in the new diagram. This is followed by drawing the graph in the new diagram, again using information about the layout in the existing diagram. These two stages are illustrated in Figure 4.



The original diagram

The new diagram before layout

New diagram after Stage 1

New diagram after Stage 2

Figure 4. Illustrating the two-stage dynamic drawing process

The rest of the paper is organised as follows: Section 2 discusses the background work to this paper; Section 3 details the graph-enhanced Euler diagram dynamic drawing method; Section 4 describes how the method is applied to the application area of diagrammatic reasoning with Euler diagrams; Section 5 details cases where drawing go wrong and suggests possible solutions; and finally, Section 6 gives our conclusions.

2. BACKGROUND

There is a well-established existing body of work on drawing Venn diagrams (which have all possible zones), for a comprehensive review see [16]. The task of drawing Euler diagrams is more difficult than that of drawing Venn diagrams (because there are many more possible configurations) and more useful (because a Venn diagram with many contours becomes difficult to interpret).

Only recently have we seen the publication of papers addressing the specific problem of drawing Euler diagrams (e.g.[3][6][7][8]). The general task of drawing Euler diagrams can be reduced to the simpler task of drawing atomic Euler diagrams and recombining to build a nested Euler diagram [7]. In practical applications this is a useful step which often reduces the number of contours in drawing task.

The embeddings obtainable from algorithms given in [6][7] were correct, but not aesthetically pleasing and could be hard to visually decipher. In [8] this problem was addressed by taking an embedded diagram, subject to some aesthetic criteria, and applying a hill climbing algorithm to lay out the diagram. The hill climbing process was guided by the use of various metrics to assess the quality of a drawing. In [15] graphs were superimposed upon the Euler diagrams, combining aesthetic-based hill climbing and force-based iteration to place graph nodes. Drawing graph-enhanced Euler diagrams widens the number of potential application areas for the work. Apart from diagrammatic reasoning systems, graph-enhanced Euler diagrams have been used to visualise information in file systems [2] and are applicable where graphs are extended in higraph or hypergraph systems such as software modelling [18], the visualization of networks [11], and database visualization [4].

The recent results in Euler diagram drawing form the basis of the work presented in this paper, but they all addressed the problem of drawing an Euler diagram as an isolated artefact (*static* drawing). In contrast, the work in this paper considers the problem of drawing a graph-enhanced Euler diagram in the context of other drawings (*dynamic* drawing).

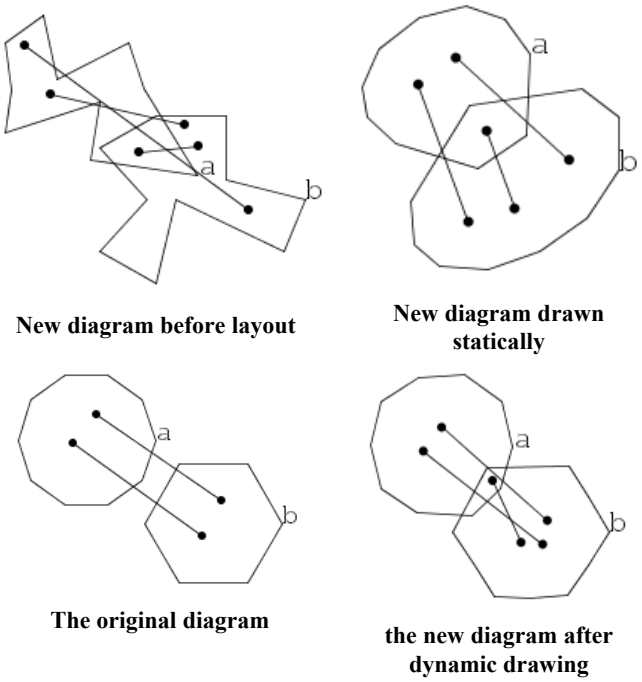
The field of dynamic graph drawing investigates the process of changing the drawing of a graph as changes are made to the underlying structure of the graph. Current dynamic graph drawing techniques, see [1] and [14] for surveys, are usually based on dynamic variations of current graph drawing methods. This work informs our choice of title for this paper, where we imagine a drawn graph-

enhanced Euler diagram as a context for the task of drawing a second diagram, related to the first but with altered contours, zones and/or graph.

A major issue when dynamically drawing a graph is to avoid disturbing the users mental map of the graph [5], a concept which relates to minimising the disruption to the current drawing, because the user has invested time in understanding the current structure. Similarly, in dynamic Euler diagram drawing we strive to present the new drawing so that it looks similar to the other diagram, to maximise the chances that a user could transfer their understanding of one diagram in interpreting the other. Any differences between the diagrams should become visually obvious and commonalities should be clearly comparable.

Figure 5 contrasts between static and dynamic graph drawing. In this figure, the static drawing approach successfully changes the new diagram into an approximation of the original. It lays the contours our nicely whilst maintaining the Euler diagram structure (the same zone set). Also the graphs are drawn to separate graph nodes, to keep nodes away from contours and to minimise edge crossings.

The third diagram in the figure is a different, contextual diagram, for illustrating dynamic drawing. It has fewer zones and fewer graph components. The fourth diagram is an attempt to draw the "new diagram" to look similar to the "original diagram". It, again, maintains the correct zone set and lays out the contours. We can see that the intersection zone in the dynamically drawn diagram is smaller than in the statically drawn diagram, because that zone is missing from the context. There are conflicting aims – to draw similar graph components in similar places, and to minimise edge-crossings. In this case, the task of highlighting commonality between the graph components has resulted in a drawing which has edge-crossings. Different weightings can be specified to achieve a balance between such conflicting aims.



New diagram before layout

New diagram drawn statically

The original diagram

the new diagram after dynamic drawing

Figure 5. Static and dynamic diagram drawing

This work on dynamic graph-enhanced Euler diagram drawing has been applied to spider diagrams [12]. Inspired by the widespread use of diagrammatic notations for modelling 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.

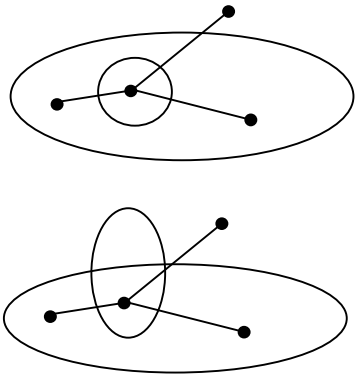


Figure 6. Two equivalent hypergraph drawings which are different when interpreted as spider diagrams.

Spider diagrams are a subset of *constraint diagrams* [17], with a restricted notation and restricted rule system. *Unitary 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. Thus the absence of a zone from the diagram conveys information, and the two diagrams in Figure 6 have different semantics.

Each spider drawn on a spider diagram 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 that are in it, and no more. Shading places an upper limit on the cardinality of sets. See Figure 7 for an example of a spider diagram with its abstract syntax and semantics.

Abstract syntax:

Contours : {*a*, *b*}

Zones: {{},{*a*},{*b*}}

Shading:{{*a*}}

Spiders : {{{},{*b*}},{{*a*}}}

Semantics:

$|A| = 1$ and $A \cap B = \{\}$ and |U-A| ≥ 1

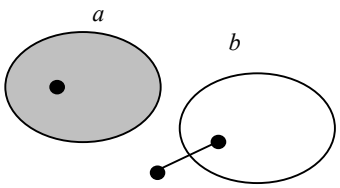


Figure 7. An abstract spider diagram and a corresponding drawn spider diagram.

The semantics of spider diagrams provide a foundation upon which we build reasoning rules. In the case of spider diagrams, it is standard to allow seven rules to transform a spider diagram into another (these rules are given in e.g. [9]). For example, one rule transforms a diagram with an absent zone into the equivalent diagram which contains the zone, 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 8. The same proof is shown again later in Figure 20, Figure 21 and Figure 22.

The seven reasoning rules each make a small change to a diagram, and they have each been proven to be valid: if a rule transforms diagram d_1 into diagram d_2 then d_2 represents a semantic consequence of d_1. Other rules could be devised which are valid, and in any logic system, the choice of rules is to some extent arbitrary.

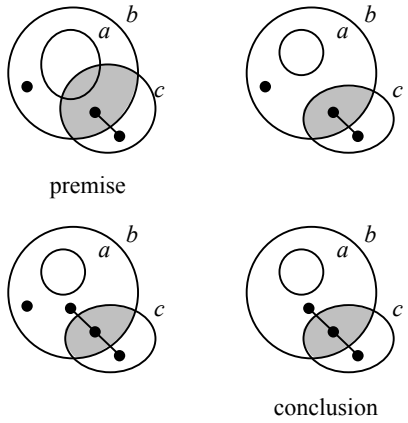


Figure 8. An example of a proof in the spider diagram reasoning system

The spider diagram reasoning system provides an ideal application for the dynamic drawing of diagrams, because we have a software tool which generates proofs of spider diagram theorems, but only generates the underlying abstract diagram sequence. Each diagram in the proof needs to be laid out for the user so that the changes that have been applied by the reasoning rules are visually clear. The first diagram can be laid out statically and the remaining diagrams dynamically drawn with the preceding diagram as context.

3. DYNAMIC DRAWING METHOD

We have implemented a two stage system for dynamic drawing of Euler diagrams enhanced with graphs that builds on the static drawing method described in [15]. Firstly we lay out the underlying Euler structure in the new diagram, using information about the layout in the original diagram. This is followed by drawing the graph in the new diagram, again using information about the layout in the original diagram. These two stages were illustrated in Figure 4.

Our strategy for dynamic layout is to draw a new diagram in a similar manner to an existing diagram, but to also include some notion of aesthetics into the new layout. There are three issues we deal with: mapping diagram items in one diagram to items in the other another, which is relatively easy for the contours and zones of an Euler diagram, as the contours in both diagrams must be uniquely labelled, but is harder for the embedded graph as it is unlabelled; the second issue is to lay out the items in the new diagram in a similar way to mapped items of the

existing diagram; the third issue is to include aesthetics into the layout of the new diagram, so that unmapped items are not drawn badly, and so that mapped items are not forced into bad layouts because of changes in the vicinity of the item.

3.1 Dynamic Euler Diagram Layout

The layout of the Euler Diagram uses a multi-criteria optimiser that integrates two specialist dynamic metrics with existing metrics that improve the general aesthetics of a diagram. These general metrics are used in addition to the dynamic metrics because of the incomplete nature of the mapping between the original and new diagrams. Where contours are not present, simply following the current layout is not possible. Where zones have been altered it may also be that the best possible match for the current layout results in a very poorly laid out diagram. The optimiser is a hill-climber, which attempts to minimise a weighted sum of the metrics.

The existing static diagram metrics are taken from the static Euler diagram layout method described in [8]. As with the previous work, contours are represented as polygons with an arbitrary number of points. This allows the use of standard algorithms to produce the metrics. There are seven single diagram metrics used. Two improve the roundness of contours: *ContourRoundnessAngles*, which balances out the angles at the points of polygons and *ContourRoundnessEdgeLength*, which balances out the length of the line segments of contours. *DiagramArea* measures the total area occupied by the diagram, and so prevents disconnected contours from moving too far apart. *ContourArea* balances out the areas of contours. *ZoneArea* balances out the areas of zones. Two metrics measure the closeness of contours: *ContourClosenessPts* uses distances between points on the contours and *ContourClosenessEdgePt*, measures the closeness of points in one contour to the line segments in the other. All of these metrics except *DiagramArea* are invariant under scaling.

To lay out static diagrams, these metrics are used in a hill climber. This moves the points of contours, and checks if the weighted sum of the metrics had been improved. If there is an improvement, the move is kept, otherwise it is discarded. As well as single points entire contours were also moved. A cooling schedule is applied in order to reduce the amount of movement as the iterations continued.

The hill climber was modified for the drawing of dynamic Euler diagrams. In particular it was clear that the dynamic metrics were each affected by either point movement or contour movement, but not both. To take advantage of this, and improve the time taken to get to a minima, the dynamic metrics were designed so that they could register for a particular movement type. This did not have an impact on the static diagram metrics, as they are all affected by both types of movement

This change has a significant consequence on measuring fitness. It means that there cannot be a global fitness function, only local fitness functions for point movement and contour movement. It is conceivable that one movement may reduce one fitness measure but in doing so increase another, however, because the functions share many metrics (all the single diagram metrics), the net overall effect is a downwards movement of both fitness functions. It is likely that having competing fitness functions would become problematic if the sets of metrics used in each have fewer metrics in common.

The motivation for the new dynamic metrics is to ensure that the new diagram looks as close as possible to the original. There are two components to this. Firstly, the position of contours that appear in both diagrams should be similar. This is implemented by the *ContourPositionComparison* metric. Secondly, the shape of two contours that appear in both diagrams should be similar. This is implemented by the *ContourPointsDifferenceComparision* metric.

ContourPositionComparison sums the square of the position differences between mapped contours. To ensure the metric does not change when the diagrams are scaled, it is divided by a value based on the area of the original diagram (as this diagram does not change, this value is a constant throughout the drawing process). There are various possible scaling values, but we use the sum of the areas of the contours in the still diagram, which relates directly to the scale of the original diagram (and so the new diagram, as it will be drawn similarly to the original) and is fairly simple to calculate. More precisely the formulae for this metric is

$$\frac{\sum_{\text{contour } C} \left(dist(C_{original}, C_{new})\right)^2}{S}$$

where $dist(C_{original}, C_{new})$ is the distance between the centres of the bounding box of contour C in the original and new diagrams. S is the scaling value calculated from the original diagram. This metric is largely concerned with the position of contours. The movement of points has only a minimal impact on its value, and so the metric is only registered for the contour movement element of the hill climber.

ContourPointsDifferenceComparision is designed to make the shape of the contour in the new diagram similar to the shape of the mapped contour in the original diagram. It works by initially finding a shift factor to lay one contour on top of another, equalising the centres of the bounding

boxes. The metric penalises points of the new contour that are distant from the mapped point in the existing contour. The output is the sum of squared differences of the mapped points of each mapped contour.

To calculate this value, the point sequences of the two contours that are to be compared need to be mapped. Consider three aspects in turn.

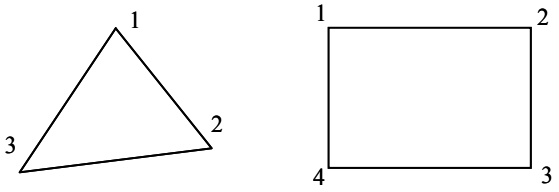


Figure 9 Contours with unequal number of points

Initially, to allow for proper points comparison the number of points in each contour need to be equalized. The number of points in the polygons representing each contour may vary in both automatically generated contours or manually created ones, see Figure 9. The numbers of points are equalized by placing new points half way along line segments of the contour with least points.

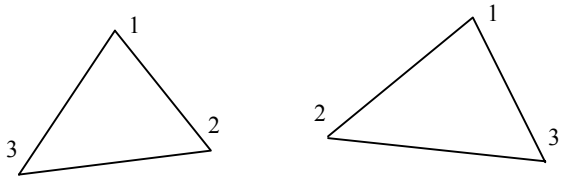


Figure 10. Contours winding in opposite orientations

Next, require that the point sequences of the contours need to be winding in the same direction - either clockwise or counter-clockwise, see Figure 10, else the contour would need to be inverted by the optimising process. To deal with this a test is made, and if they are in different directions, one point sequence is inverted.

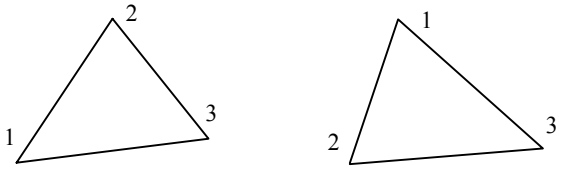


Figure 11. Contours with rotated points

Finally, the points in the sequences are aligned so that the mapped points are close together, otherwise the optimiser would need to rotate the points, see Figure 11. This is done by finding the two closest points of each contour, and translating each point labelling to start the point sequence with the closest points.

More precisely *ContourPointsDifferenceComparision* is:

$$\frac{\sum_{contour\ C}\left(\sum_{\substack{Eoriginal\ in\ Coriginal,\\ Enew\ in\ Cnew}}\left(dist(E_{original},E_{new})\right)^2\right)}{S}$$

where $dist(E_{original},E_{new})$ is the distance between the mapped points in the original and new contours in the new diagrams. S is the same scaling value as used for *ContourPositionComparison*. This metric is concerned with the shape of contours, and the movement of contours does not change its value, hence is it only registered for the point movement element of the hill climber.

There is an alternative measure of the difference in contour shape, and that is to find the difference between the polygons that represent the two contours, and attempt to minimise the area of the difference. The difficultly with this is that very thin polygons have little area, and are difficult to reduce, but have a very noticeable impact on the drawing. When this metric was tried, the optimiser tended to reduce the difference between the polygons by making these thin polygons, rather than exactly equalizing the contours.

3.2 Dynamic Embedded Graph Layout

The work in the previous section on Dynamic Euler Diagram Layout results in a pair of drawn Euler diagrams which look similar. Work described in [15] allow us to place the graph superimposed upon the Euler diagram so that each graph node belongs to the correct Euler diagram zone. If we use this algorithm to draw the graphs, they are nicely drawn, but a graph that is identical in the new and original diagrams can appear very differently in each. Hence, to create a dynamic drawing, we use the placing of the graph nodes in the original diagram to inform the placing of the graph nodes in the new diagram.

3.2.1 Mapping

To work out which original nodes' positions should affect which new nodes' placing, we have to find a mapping (a partial injective function) from the nodes of the original diagram to nodes in the new diagram. For some nodes in the new graph, the mapping associates (unique) nodes in the original graph. If the two graphs are identical, the mapping will be a one-to-one mapping between the graph nodes, but in general there will be some nodes in the original graph unmapped and some modes in the new graph unmapped.

The simplest mapping considers connected components of the graphs in the two diagrams. For each component in the original graph, an isomorphic graph is sought which shares the same habitat (in practise, the graph components are simple and the search for isomorphic components is assisted by node assignment to zones). If such an isomorphic graph is found, then its nodes are mapped with the nodes in the new graph. An example is shown in Figure 12, where the mapping is shown using node labels. In this figure the unlabelled nodes do not participate in the mapping.

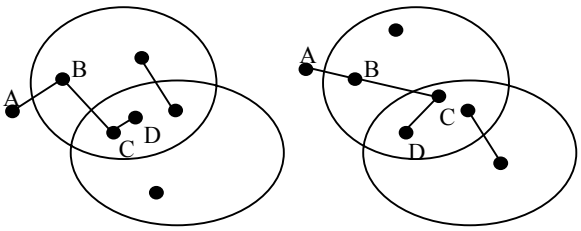


Figure 12. Simple mapping between graph nodes

A more sophisticated mapping would seek components which are "nearly" isomorphic – perhaps components which differ by a single node. This sort of partial matching involves a difficult problem of choosing which components to map if there are multiple "similar" graph components. Some examples are purely symmetrical, and an arbitrary decision could be made. In Figure 13, we could equally map A1, B1 to A and B, or we could equally map A2 and B2 to A and B.

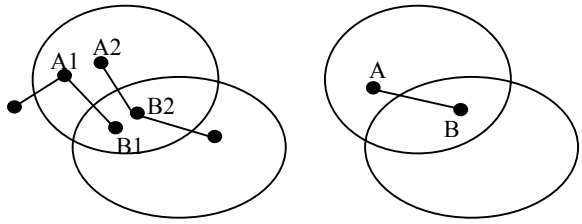


Figure 13. Symmetry and mapping graph nodes

In other cases choices would have to be made more carefully to maximise the mapping's ability to draw similar diagrams. In Figure 14, simple pairwise consideration between the CDE component on the left and the AB component on the right may lead to a mapping between D and A and between E and B. However, this would miss the opportunity to use the other component, CD, in the right hand diagram. In this way, seemingly arbitrary decisions about mapping components may have wider repercussions about mapping choices elsewhere in the diagrams. For this reason, we have avoided implementing anything more sophisticated than the simple node mapper illustrated in Figure 12.

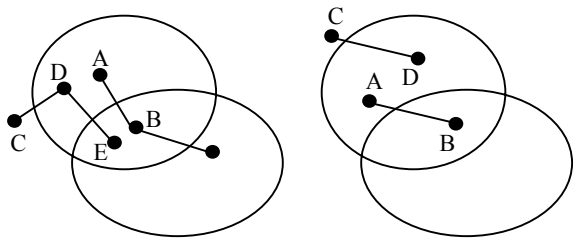


Figure 14. Problems in mapping between graph nodes

The mapping between nodes of the graphs is used to determine some of the forces exerted on graph nodes, as described in the next section. Nodes which are associated under the mapping are encouraged to be drawn in similar positions (relative to the zone they are in) but are also subject to other forces.

3.2.2 Force model

The force model for laying out the nodes in the new diagram is based on the version used for laying out static embedded graphs as described in [15]. The forces are adjusted so that nodes which have a mapped node in the original diagram are encouraged to move closer to the corresponding position in the original diagram. However, changes to the structure of the Euler diagram may mean that the corresponding position is undesirable, or worse, it is outside the correct zone. Hence, the force system uses the forces from the static method to ensure that the layout does not have very poor aesthetics and that the diagram retains its structure. Nodes that do not have a mapped node in the original diagram are, in effect, placed with the standard static method.

A node belonging to a particular zone must first be placed such that it is contained within the region defined by the zone in the new diagram. As with the static method we place nodes randomly by first drawing a horizontal line through the containing zone that meets the zone at a random point. The node is then placed on this line. This strategy is designed for quick placements of nodes in the correct zone. The application of the force model which then follows will place nodes in aesthetically pleasing positions, and if a node has a corresponding mapped node, it will be placed close to the relevant location found from the original diagram.

After initial placement, refinement of node locations is achieved by applying a force model to the set *M* of nodes in the zone. As with the static method we have a repulsive force acting between each pair of nodes in the zone, which separates nodes evenly and a repulsive force between nodes and line segments, which prevent nodes escaping from a zone or getting undesirably close to the boundary of a zone. In addition, the dynamic method includes a force that attracts nodes to the location of nodes they are mapped

to in the original diagram, so encouraging the layout to be similar to the original where appropriate.

The repulsive force between nodes is based on that of force model by Fruchterman and Reingold [10] and is inversely proportional to the separation d, and proportional to the number of nodes, $|M|$, in the zone. A constant c is used to affect the desired separation between pairs of nodes. The repulsive force between two nodes is given by $|M| \times \frac{c}{d}$.

The repulsive force between nodes and line segments acts on the nodes only; it does not move the line segments. It is proportional to $|M|^2$, as this helps to contain larger sets of nodes where there will be more node-node repulsions. As the zone may consist of an arbitrary number of line segments of arbitrary lengths, the repulsive force is also proportional to the length of the line segment l. The repulsive force between a line segment and a node is given by $|M|^2 \times \frac{lc}{d^2}$.

The attractive force is applied to each node that has a mapping to a node in the original diagram. The force acts towards the position of the mapped node, encouraging the graph in the new diagram to be laid out similarly to the graph in the original diagram. The magnitude of the force applied to the node is directly proportional to the distance to the mapped node squared. k represents a constant that can be used to adjust this attractive force in relation to the two repulsive forces. The attractive force towards mapped node position is given by $\frac{kd^2}{c}$.

The application 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, the repulsive forces from all other nodes in the same zone and possibly the force towards the corresponding node in the still diagram. 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, however we cap the maximum value of the movement to prevent very strong forces from moving nodes a long way and therefore possibly breaking the structure of the diagram. After a number of iterations, the system nears an equilibrium and the nodes occupy their new locations.

With three different types of forces acting simultaneously, some care must be taken to choose suitable values for each parameter. Our experimental framework uses $c = 2$ and $k = 5\text{x}10^{-6}$. The purpose of the repulsive force exerted on nodes by line segments is to prevent nodes escaping from their containing zone. If the attractive force towards a node's position in the original diagram is made too strong, the resultant force acting on the node could cause it to escape from the zone. For this reason, it is important to choose a suitable value for k, which typically leads to a compromise between preserving structural correctness (which is essential) and preserving the mental map of the user.

4. EXAMPLE – DIAGRAM PROOF SEQUENCES

In the section we demonstrate the dynamic drawing as applied to some spider diagram proofs: sequences of spider diagrams. Firstly we discuss the alterations needed to adapt our method to this specific application area and then we give some detailed examples.

4.1 Extra steps used for this application

As discussed in [15], the graphs for spider diagrams are unusual in that the abstract syntax specifies only the connected components of the graph (the habitats of the spiders). The graph edges serve only to link together connected components into trees, and any tree would suffice to convey the same diagram semantics. When drawing a static diagram, we collected together relevant graph nodes and drew an arbitrary spanning tree for that node set.

When drawing spider diagrams dynamically, the node mapping described in section 3.2.1 may associate a spider in the original diagram with one (sharing the same habitat) in the new diagram. The force model will strive to present the nodes of these two spiders in similar positions so that the spiders are recognisably "the same" spider. However, unless attention is paid to the chosen edges, two spiders could still end up looking different, as illustrated in Figure 15.

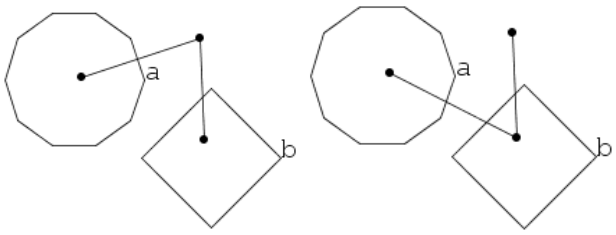


Figure 15. The importance of relocating edges in spider diagrams

The mapping between nodes of one spider and nodes of another can be used to choose which edges should be chosen to build the spanning tree in the new diagram, and this step is taken before the force model is applied. An example of reallocation of edges is illustrated in Figure 16.

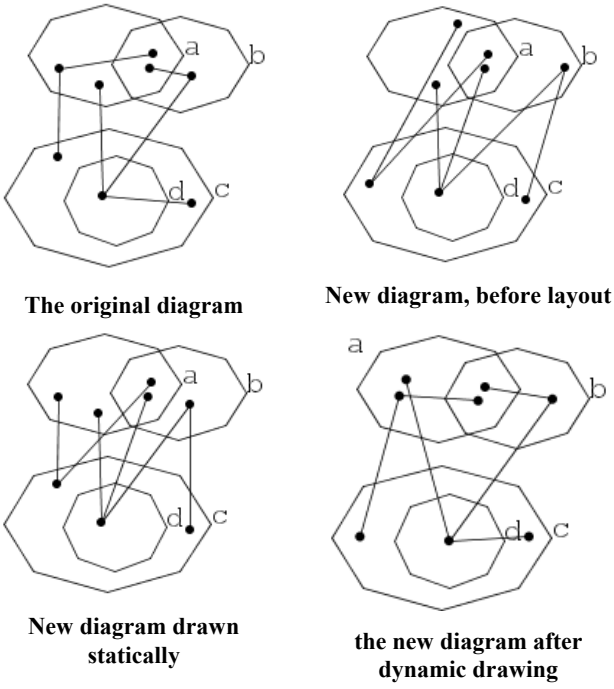


Figure 16. Reassigning edges using the contextual diagram

Another change is the position-exchanging step that was used in the static case. To recap, if a diagram had n nodes in zone z, the static drawing algorithm first identified n suitable node positions, then allocated nodes to different positions, using metrics to determine whether exchanging positions gave an improvement or not. The metrics penalised diagrams with edge-crossings and diagrams whose total edge-length was large. In the dynamic case, it would be a backwards step to change the position of a node whose position had been moved using the force model to match the position of a partner node in the original diagram. The position-exchanging algorithm is still used but only nodes which weren't in the mapping between diagrams participate. This is illustrated in Figure 17. In the static drawn diagram, the three positions have been distributed within the zones, then positions exchanged to minimise edge crossings. However, in the dynamically drawn diagram, an attempt has been made to mimic the context diagram, and we would expect two of the diagram components to include an edge-crossing. The third diagram component also involves edge-crossings but it has no other (non-mapped) node positions to exchange places with to reduce the complexity of the diagram.

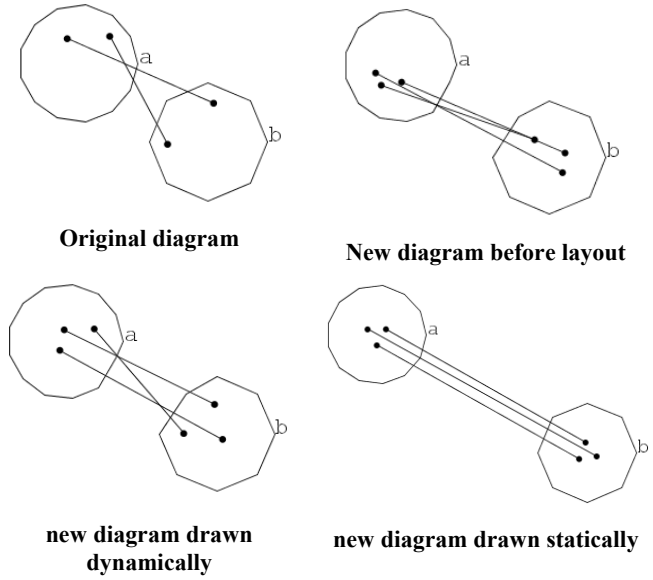
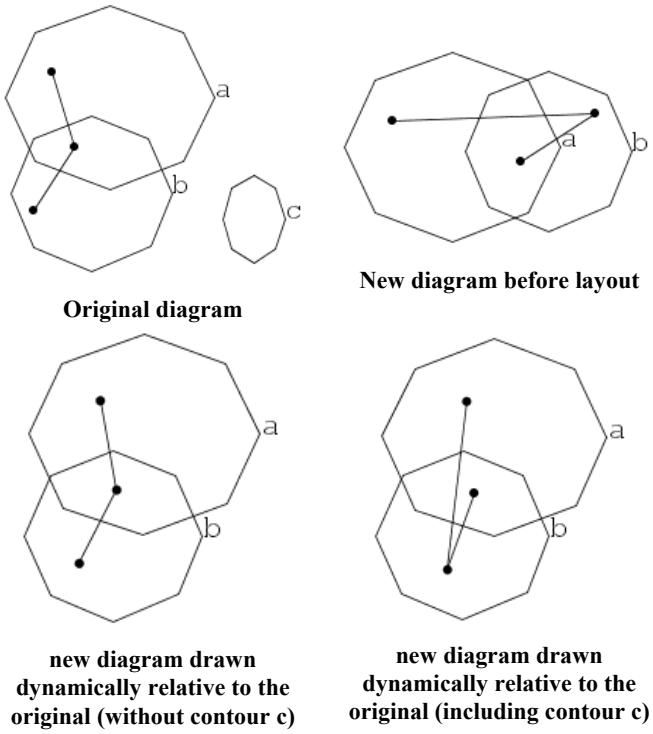


Figure 17 Exchanging node positions

If the underlying Euler diagram changes its contour set, then the zones become different abstract zones (the abstract zones are identified by the set of containing contours and the set of excluding contours). If a contour is removed from a diagram, even if it seems to leave part of the diagram unaffected, the graph components will not be mapped across, and the graphs will be drawn independently (see, e.g. Figure 18). This failing to match graph components could be fixed by using the notion of *corresponding regions* in Euler diagrams, see [13].



Original diagram

New diagram before layout

new diagram drawn dynamically relative to the original (without contour c)

new diagram drawn dynamically relative to the original (including contour c)

Figure 18. The effects of changing the contour set on graph component matching

4.2 Examples

In this section we show the method working on some example proof sequences. The first example shows several different layouts for the same proof. The following two examples are briefer, and compare the original undrawn lay out of the proofs against the drawn version.

To make the examples consistent they have all been drawn with the same parameters for Euler diagram multi-criteria optimiser and for the graph force algorithm. This inevitably produces a compromise, and better results for individual examples could have been improved by tuning the numbers.

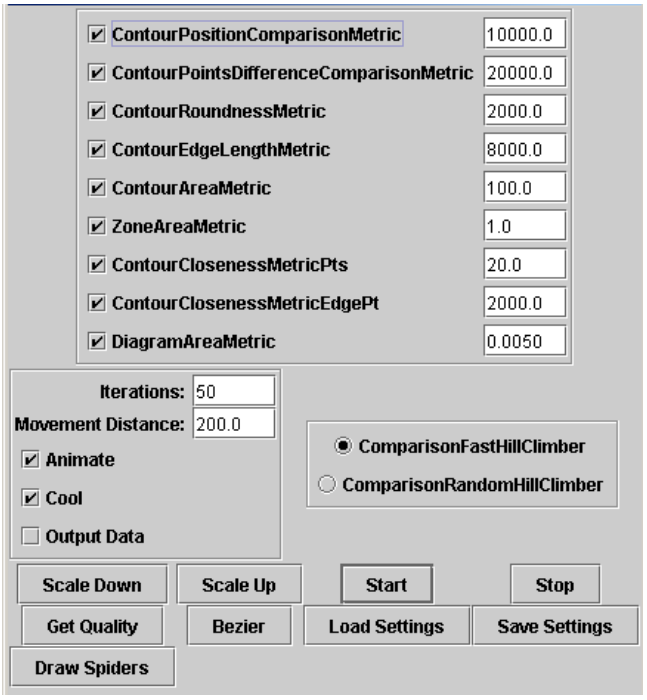


Figure 19. Criteria weights in the control window

The metrics and their weights for the Euler diagram optimiser are given in Figure 19, which also shows the control window for the dynamic drawing. As with most multi-criteria systems the weights serve two purposes, to define the importance of the metrics and to normalize the values of the metrics, which may return values in very different ranges. In this

For the diagram at the top of Figure 24 the values for these metrics, including the above weights is:

ContourPositionComparison	634.8
ContourPointsDifferenceComparison	82592
ContourRoundness	0.00031
ContourEdgeLength	0.00028
ContourArea	18.02
ZoneArea	5.337
ContourClosenessPts	2148.7
ContourClosenessEdgePt	52.15
DiagramArea	0.02296

It can be seen that the two dynamic metrics at the top are given much higher priorities than the standard static metrics. This is to ensure diagrams are drawn similarly. The next highest importance is given to the closeness metrics, to counter the tendency of the similarity metrics to

push contours against the border of other contours when the mapped contour is not in a position that can be closely copied. The other metrics are quite low in this diagram. This is normal, but a metric will spike highly, and thereby become more important if the drawing is very poor with regard to it.

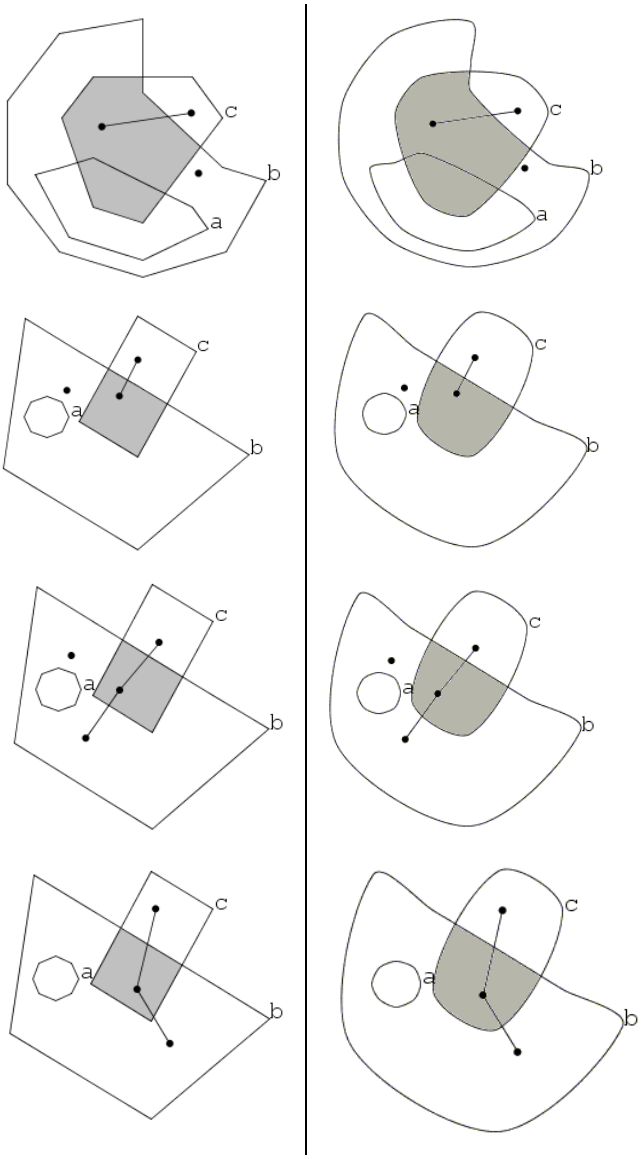


Figure 20. A proof sequence that has not been laid out, only placed with the standard embedder. The diagram on the right is the same as the diagram on the left, except for rounding of edges using a Bezier method.

Figure 20, Figure 21 and Figure 22 show an extended example of drawing a diagram proof sequence with the dynamic method. This sequence is that shown in Figure 8, and allows the reader to compare the automatic dynamic approach described in Section 3 against an "ideal" hand drawn layout. The right hand side of each of these figures show the same diagram layout as the left hand side, but with a Bezier rounding method applied to the edges. Each of the two figures that use the dynamic method, Figure 21 and Figure 22 use a different starting mechanism.

Figure 20 shows the simple initial layout with no attempt at drawing the diagrams nicely. These Euler diagrams are drawn with the method described in [6], and the graphs are laid out randomly, using the method described in Section 3.2.2. This is the initial position for all the diagrams in this paper (except for those which are hand drawn). The layout method then alters this initial position to a more aesthetically pleasing one.

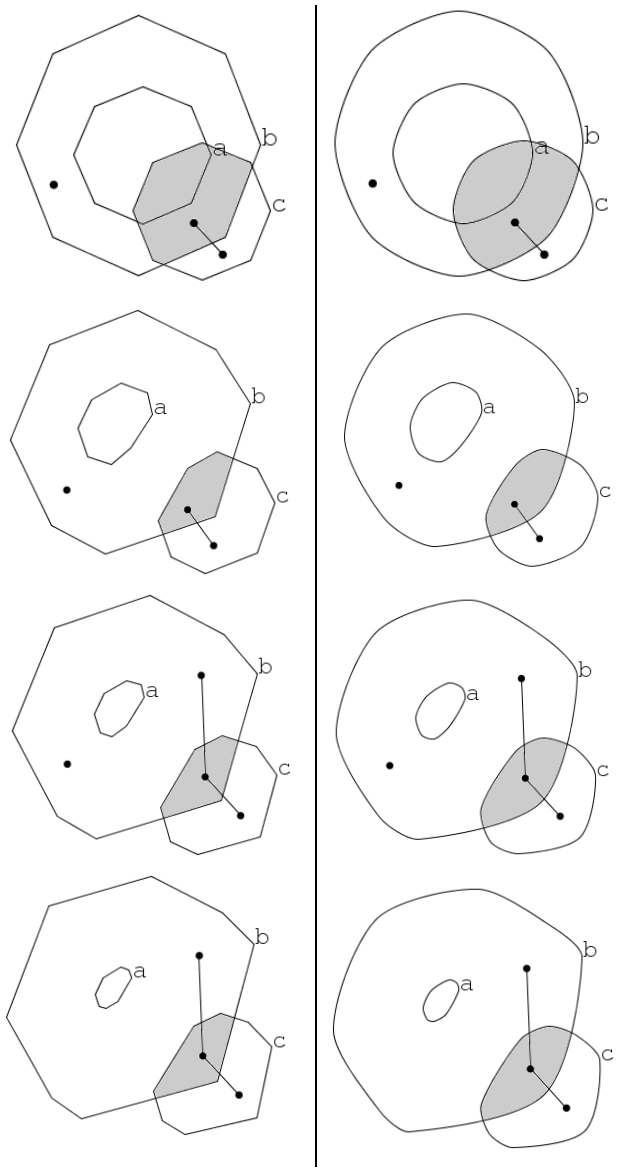


Figure 21. A proof sequence for which the initial diagram has been hand drawn, and subsequent diagrams have been drawn by the dynamic method. The diagrams on the right are the same as on the left, but improved with a Beziering method

Figure 21 shows a dynamic sequence for which the initial diagram is hand drawn subsequent diagrams are laid out using the dynamic drawing method, as with the following examples, the dynamic drawing method is applied by taking the previous diagram in the sequence as the original, rather than always using the first diagram. Both the Euler contours and graphs are maintained in relatively similar positions to the previous diagram, even after structural changes are made, we regard the layout of this sequence as successful.

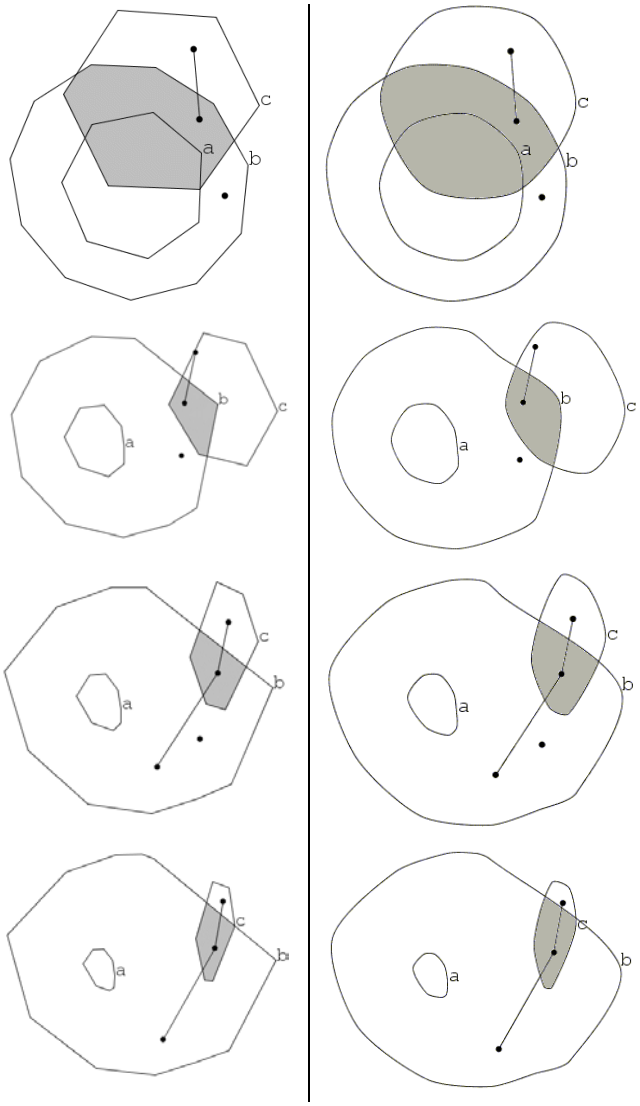


Figure 22. A proof sequence for which the initial diagram has been drawn with the static drawing method, and subsequent diagrams have been drawn by the dynamic method. The diagrams on the right are the same as on the left, but improved with a Beziering method

Figure 22 shows a proof sequence where the first diagram is laid out using the static drawing method described in [15]. This sequence is still aesthetically acceptable, but minor problems in early layout are compounded in later diagrams. In particular the high weightings given to the contour separation Euler criteria are not appropriate for this example. A set of weightings used in a static context would be more appropriate here.

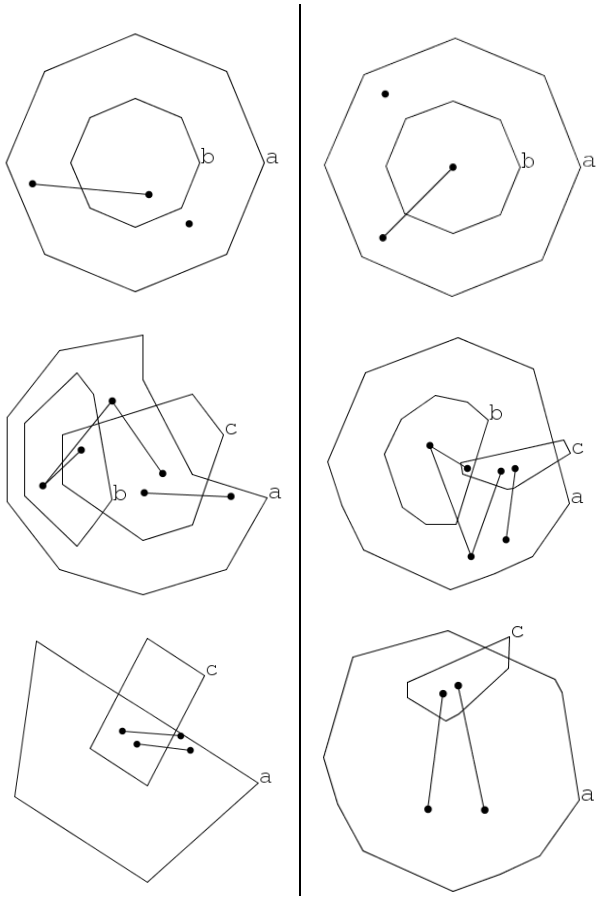


Figure 23. A proof sequence with the no layout method applied to the diagram on the left. The static drawing method has been applied on the first diagram on the right, with the dynamic method applied subsequently.

Figure 23 shows a more dynamic sequence, in each diagram varied changes are being made to both the Euler diagram and graph. The initial layout is shown on the left. This changes the layout of the diagram radically for each step in the sequence. The right hand side shows an automatically laid out diagram at the top, followed by applications of the dynamic method below it. The contours retain approximately their correct positions, although some difference in graph layout can be discerned. The graph from the first to the second diagram is particularly distinct, because our simple mapping method does not connect the two graphs, and they are laid out independently.

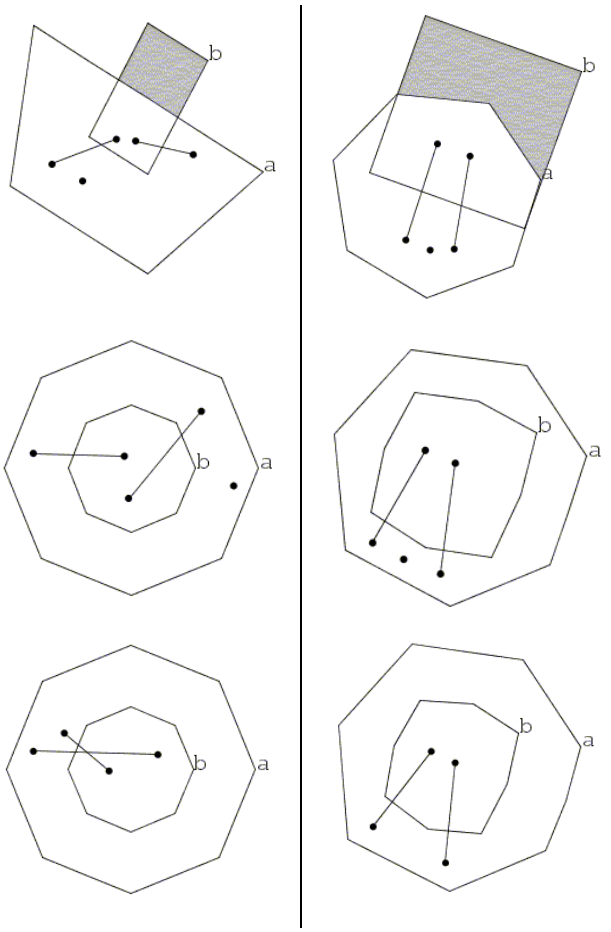
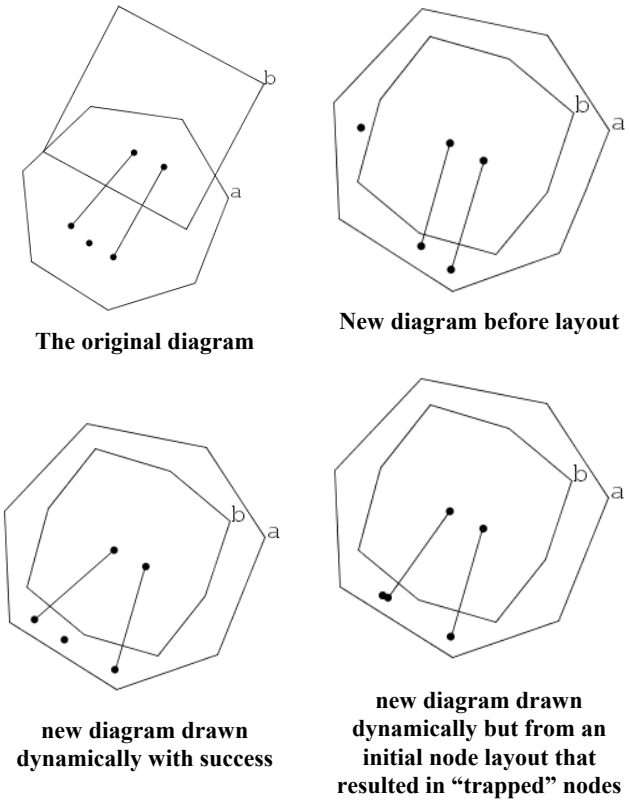


Figure 24. A proof sequence with the no layout method applied to the diagram on the left. The static drawing method has been applied on the first diagram on the right, with the dynamic method applied subsequently.

Figure 24 shows a good example of a graph structure being maintained by the method. The graph layout on the right hand side is clearly repeated. The initial layout for the lower two diagrams on the left has a fortuitously good embedding in this case. On the right, the changes in Euler diagram structure do not completely break the relationship, between the Euler diagrams in the first and second of the sequence, with the shape of the square in the top being reflected below.

5. FURTHER WORK

Whilst our method is usually effective, there are some problematic cases of diagram. In this section we comment on some of the situations where the current drawing system can reach poor layouts and discuss possible solutions.



The original diagram

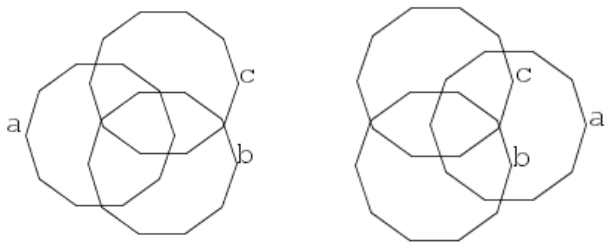
New diagram before layout

new diagram drawn dynamically with success

new diagram drawn dynamically but from an initial node layout that resulted in "trapped" nodes

Figure 25. An example of conflict between three forces

Three different forces are applied during the simulation of the force model. More often than not, these interact to produce desirable results. However, there are some cases where these forces can be seen to conflict with each other. The example in Figure 25 illustrates one such problem. The bottom right diagram has had the force model applied to it from a random initial layout of nodes. The lone node has become trapped on the wrong side of the node it is nearest to. Both nodes are relatively close to the zone boundaries that form a valley along which the nodes prefer to move. The two nodes are unable to pass each other along this valley because the repulsive force between these two nodes is countering the attractive force towards their partnering nodes in the original diagram. This type of problem can usually be solved by reapplying the initial random layout and force model. An alternative would be to use the mapping information about nodes to find a more suitable initial layout for the parts of the graph which have this information available.



Original diagram **New diagram**

Figure 26. In the new diagram, contour a cannot move to the desired position on the other side of b and c

The hill climber used as our optimiser can reach local minima, particularly when placing contours that are far away from their desired position, with other contours in the path between current and desired position, see Figure 26. Other more sophisticated optimisers, such as simulated annealing or genetic algorithms, could be applied, but would take longer to run. Another approach to solving problems of this sort is to modify the movement method. At the moment there are two sorts of movement, contour points and the contours themselves, neither of which are not moved very far in a single step. It would be possible to make wider movements, directed by some heuristic, or to move larger collections of contours.

6. CONCLUSIONS

We have developed a dynamic drawing method for Euler diagrams enhanced with graphs, which builds on a static drawing method. It firstly draws the Euler diagram of the new diagram like the Euler diagram of the original using a multi-criteria approach. The embedded graph of the new diagram is then drawn like the original with a force based method. The drawing method also incorporates aesthetic notions so that where parts of the new diagram are different they can be drawn nicely. The method works effectively and we have shown it being applied to visualizing diagram proof sequences.

We consider this work to be extendable beyond dynamic drawing to general example based drawing. Users could teach a tool how to automatically lay out diagrams. A library of existing diagrams would be consulted before a new diagram is drawn. A challenge for this method includes deciding which diagram would be chosen to form the pattern, by developing a difference measurement between diagrams.

We use aesthetic criteria to draw the Euler diagram. This method is flexible, allowing changes in weights and different criteria according to user preference. The force model used to draw the graph also allows for tuning of different layout preferences to a lesser extent. However, some weaknesses of our method could be addressed. Firstly we have a two stage process which firstly changes the Euler diagram and then the contained graph. In many applications it would be more desirable to combine the drawing so that there is a compromise between the layout of the Euler diagram and the graph. This could be achieved, at the cost of execution time, by laying out the graph using a multi-critieria optimiser. This could then be directly integrated into the Euler diagram optimiser, and weights assigned for the desired compromise.

The method presented here lays out one diagram based on the current layout of another. For applications such as user exploration of proofs this is the best strategy, as the user has a mental map of the first diagram. However, to get best compromise for all drawings in a sequence of diagrams which are already generated, alternative methods might be employed. An example application is presenting proofs where all the diagrams in a sequence have already been generated. To achieve this a set of metrics measuring the fitness of the drawings across all diagrams could be developed, and employed on the sequence. The metrics could for the most part be variations on those used in the current system.

7. ACKNOWLEDGMENTS

This work has been partially supported by EPSRC grants GR/R63509/01 and GR/R63516/01.

8. REFERENCES

[1] Battista G., Eades P., Tamassia R. and Tollis I.. Graph Drawing: Algorithms for the Visualisation of Graphs. Prentice Hall. 1999.

[2] De Chiara R., Erra U. and Scarano V. VENNFS: A Venn-Diagram File Manager. Proc. IEEE Information Visualization (IV03). pp. 120-126. 2003.

[3] Chow S. and Ruskey F. Drawing Area-Proportional Venn and Euler Diagrams. To appear in Proccedings of GD2003. LNCS. Springer Verlag.

[4] Consens M.P. and Mendelzon A.O. Hy+: A Hygraph-based Query and Visualization System. In Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp. 511-516, 1993.

[5] Eades P., Wei Lai, Misue K., and Sugiyama K. Layout Adjustment and the Mental Map, *Journal of Visual Languages and Computing* 6, (1995), 183 - 210.

[6] Flower J., and Howse J. Generating Euler Diagrams, *Proc. Diagrams 2002* LNAI 2317, Springer Verlag, pp. 61-75. 2002.

[7] Flower J., Howse J. and Taylor J. Nesting in Euler Diagrams: syntax, semantics and construction. *Journal of Software and Systems modelling*, issue 1, article 7, Springer Verlag. 2003.

[8] Flower J., Rodgers P. and Mutton P. Layout Metrics for Euler Diagrams. *Proc. 7th IEEE Information Visualization (IV03)*. pp. 272-280. 2003.

[9] Flower J., and Stapleton G.. Automated Theorem Proving with Spider Diagrams. *To appear in proc. Computing Australasian Theory Symposium (CATS04)*.

[10] Fruchterman T.M.J. and Reingold E.M. Graph Drawing by Force-directed Placement. Software – Practice and Experience Vol 21(11). pp. 1129-1164. 1991.

[11] GXL web page: http://www.gupro.de/GXL/examples/hypergraphNav.html.

[12] Howse J., Molina F., Taylor J., Kent S. and Gil J. Spider Diagrams: A Diagrammatic Reasoning System, Journal of Visual Languages and Computing 12, 299-324. 2001

[13] Howse J., Stapleton G., Flower J. and Taylor J. Corresponding regions in Euler diagrams, *Proc. Diagrams 2002* LNAI 2317, Springer Verlag, pp. 146-160. 2002.

[14] Kaufmann M. and Wagner D.. Drawing Graphs: Methods and Models, LNCS 2025. 2001.

[15] Mutton, P.J., Rodgers P.J., and Flower, J.A. Drawing Graphs in Euler Diagrams. *To appear in Diagrams 2004*. LNAI, Springer-Verlag.

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

[17] Stapleton G., Howse J. and Taylor J. A constraint diagram reasoning system. *Proc. Distributed Multimedia Systems, International Conference on Visual Languages and Computing* (VLC '03). pp. 263-270, Miami, USA, 2003.

[18] Storey M.-A. D. and Mueller H.. Manipulating and documenting software structures using SHriMP views. In Int. Conf. in Software Maintenance, pp. 275-285. IEEE. 1995.