University of Kent

Kent Academic Repository

Rodgers, Peter, Mutton, Paul and Flower, Jean (2004) *Dynamic Euler Diagram Drawing.* In: Proceedings IEEE Symposium on Visual Languages and Human-Centri Computing (VL/HCC'04). . pp. 147-156. IEEE ISBN 0-7803-8696-5.

Downloaded from

https://kar.kent.ac.uk/14089/ The University of Kent's Academic Repository KAR

The version of record is available from

https://doi.org/10.1109/VLHCC.2004.21

This document version

UNSPECIFIED

DOI for this version

Licence for this version

UNSPECIFIED

Additional information

Versions of research works

Versions of Record

If this version is the version of record, it is the same as the published version available on the publisher's web site. Cite as the published version.

Author Accepted Manuscripts

If this document is identified as the Author Accepted Manuscript it is the version after peer review but before type setting, copy editing or publisher branding. Cite as Surname, Initial. (Year) 'Title of article'. To be published in *Title of Journal* , Volume and issue numbers [peer-reviewed accepted version]. Available at: DOI or URL (Accessed: date).

Enquiries

If you have questions about this document contact ResearchSupport@kent.ac.uk. Please include the URL of the record in KAR. If you believe that your, or a third party's rights have been compromised through this document please see our Take Down policy (available from https://www.kent.ac.uk/guides/kar-the-kent-academic-repository#policies).

Dynamic Euler Diagram Drawing

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

Paul Mutton
Computing Laboratory
University of Kent, UK
pjm2@kent.ac.uk

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

Abstract

In this paper we describe a method to lay out a graph enhanced Euler diagram so that it looks similar to a previously 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 can be generated from an abstract representation, its initial embedding may be very different from that of the original. We have developed comparison measures for Euler diagrams, integrated into a multicriteria optimizer, and applied a force model for associated graphs that attempts to move nodes towards their positions in the original layout. To further enhance the usability of the system, the transition between diagrams can be animated.

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 and excluded from all other contours in the diagram are called zones. In Figure 1 there are two Venn diagrams shown and in Figure 2 we have a Venn diagram and an Euler diagram. Venn diagrams contain all possible zones for the contours, whereas Euler diagrams may have missing zones. Euler diagrams are frequently mistakenly called Venn diagrams.

A graph enhanced Euler diagram is an underlying Euler diagram with a superimposed graph. 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 3.

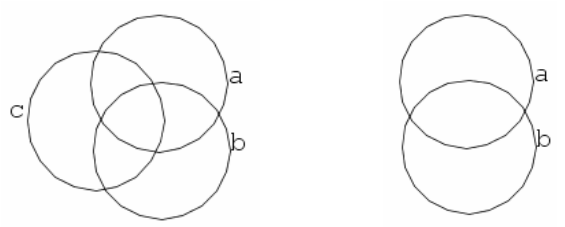
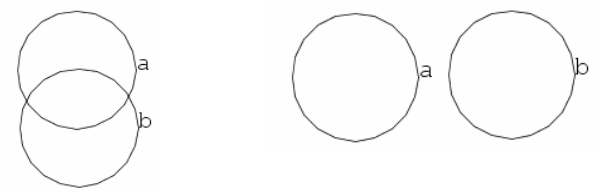


Figure 1. Diagrams with different contour sets



Venn & Euler diagram Euler diagram

Figure 2

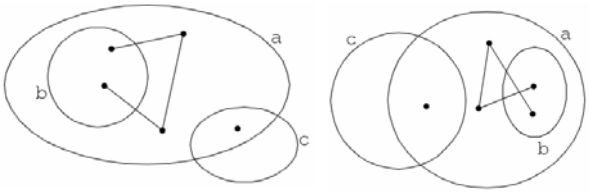


Figure 3. Two drawings of a graph enhanced Euler diagram

Graph enhanced Euler diagrams combine the immediate 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 node grouping is liable to change, because of changes to the underlying data structure, or because a user has edited the diagram.

The dynamic diagram application presented in this paper is that of laying out a sequence of diagrams that comprises a diagrammatic proof. When writing proofs with diagrams, reasoning steps transform one diagram into the next and these reasoning steps can make structural changes in the diagram. To improve 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. Possible changes to an Euler diagram include the addition or removal of contours (see Figure 1), or changes to the zone set (see Figure 2). In addition to changes to the underlying Euler diagram, the graph associated with the diagram may change with nodes or edges being deleted or added.

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.

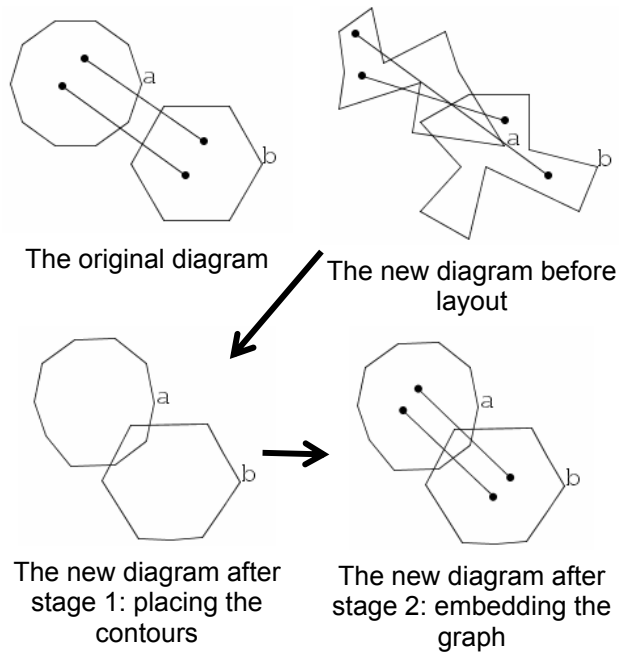


Figure 4. Illustrating the two stage dynamic drawing process

The method described in this paper draws a *new diagram* (whose layout will change during an iterative process) in the context of the layout of an *original diagram*. This original diagram is a diagram used for reference and influences the layout of the new diagram. The layout of the original diagram remains unchanged.

This work extends a static (non-dynamic) graph enhanced Euler diagram drawing method previously developed by the investigators [8],[14]. First we lay out the underlying Euler structure of the new diagram using information about the layout in the original diagram. Next we draw the graph in the new diagram, again using information about the layout in the original diagram. See Figure 4 for an example.

The iteration is guided by a multicriteria based hill climbing process. Such approaches have been widely used to solve different kinds of problem. In this paper the iteration is guided by novel metrics, making original kinds of diagram change in the iteration, and used for the first time in the application area of diagrammatic proof presentation.

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 produces poor results and suggests possible remedies; and finally, Section 6 gives our conclusions.

2. Background

There is a well established existing body of work on drawing Venn diagrams. Venn diagrams have all possible zones, unlike Euler diagrams which have a subset of zones. For a comprehensive review of Venn diagram layout see [15]. We consider Euler diagrams to be more useful because Venn diagrams with many contours become difficult to interpret, however he task of drawing Euler diagrams is more difficult than that of drawing Venn diagrams because there are many more possible configurations.

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 an Euler diagram can be reduced to the simpler task of drawing parts of the diagram and recombining to build a nested diagram [7]. In practical applications this is a useful step because drawing each part is a task involving fewer contours than the resulting diagram.

The embeddings obtainable from algorithms given in [6],[7] are correct, but are typically not very comprehensible. In [8] this problem is 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 is guided by the use of various metrics to assess the quality of a drawing. In our previous work on static layout of graph enhanced Euler diagrams [14] aesthetic based hill climbing laid out the Euler diagram and force based iteration was used to place the nodes of the graph.

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 [17], the visualization of networks [11], and database visualization [4].

These recent results in Euler diagram drawing form the basis of the work presented in this paper, but they all address 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 that is changing, and basing the layout of later versions on the layout of earlier versions (*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, see [1],[13] for surveys. This work informs the 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 different contours, zones and/or graph.

A major issue in dynamic graph drawing is to avoid disturbing the user's 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 original 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 shown.

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 [14]. 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 are illustrated in Figure 4.

Our strategy for dynamic layout is to draw a new diagram in a similar manner to an existing diagram, and to also include some notion of aesthetics into the new layout. There are three issues we deal with: the first is to map (some) diagram items in one diagram to (some) items in the other. This task 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 may be 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 to the diagram structure.

To ensure that the user's mental map of the proof sequence is as undisturbed as possible, we have a system of animating the transition between two diagrams. Firstly we fade out the items appearing in the original diagram, which are not in the new diagram, we then animate the movement of the contours and spiders to their new location, and finally we fade in the items appearing in the new diagram that were not in the original.

3.1. Dynamic Euler Diagram Layout

The method uses a multicriteria hill climbing optimizer which attempts to minimise a weighted sum of a number of metrics. It integrates two specialist dynamic metrics with eight existing static metrics that improve the general appearance of a diagram. The static metrics are used in addition to the dynamic metrics because of the incomplete nature of the mapping between the original and new diagrams. Where the contour sets of the original and new diagrams are different, 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 relative weight of the metrics were chosen and refined by experimentation, see [8] for more details about combining metrics.

3.1.1. New Dynamic Metrics. Two new dynamic metrics are used. The motivation for these is to ensure that the new diagram looks as similar as possible to the original. 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 *ContourPointsDifferenceComparison* 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 *S,* the sum of the areas of the contours in the original diagram. More precisely this metric is:

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

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.

The *ContourPointsDifferenceComparison* metric 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 shifting one contour to lay on top of another, by equalising the centres of the bounding boxes. A mapping is found between points on the 2 contours and the metric penalises mapped points which are far apart.

To find the mapping between points, we add new points half way along line segments of the contour with least points until the corresponding contours have an equal number of points. A point is found on the new contour which is closest to a point on the original contour. We use this pairing to begin indexing the points around the contours, and make sure we continue the indexing in a clockwise manner around both contours. This indexing creates a one-to-one correspondence; the required mapping between points on the two contours.

ContourPointsDifferenceComparison 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. *S* is the same scaling value as used for *ContourPositionComparison*.

Initially, an alternative, simpler metric was implemented to compare contour shapes. The metric measured the areas of the differences between contours (the symmetric difference of the interiors of the two contours). This approach was rejected because it resulted in a tendency towards contours with long, thin sections extending outside the reference contour. These extensions had small area but resulted in poor layout.

3.1.3 Existing Metrics. We use existing static metrics taken from the layout method described in [8]. The metrics *ContourRoundnessAngles* and *ContourRoundnessEdgeLength* have been designed to improve the roundness of contours. *DiagramArea* measures the total area occupied by the diagram, and is used to prevent disconnected contours from moving far apart. *ContourArea* (*ZoneArea*) balances out the areas of contours (zones). The metrics *ContourClosenessPts* and *ContourClosenessEdgePt,* prevent contours moving too close together. All of these metrics except *DiagramArea* are invariant under scaling.

3.1.2. Hill Climber. The hill climber was modified from the static version. In particular it was clear that the dynamic metrics were each affected by either point movement or contour movement, but not both. To 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 drawing metrics, as they are all affected by both types of movement.

ContourPositionComparison is largely concerned with the position of contours. The movement of points has only a minimal impact on the value of this metric, and it is only registered for the contour movement element of the hill climber.

In contrast, *ContourPointsDifferenceComparison* is concerned with the shape of contours, and the movement of contours does not change its value, hence it is only registered for the point movement element of the hill climber.

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 static 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.

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 [14] 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 isomorphic (respecting zones) in the new and original diagrams can appear very differently in each. 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. We find a mapping between some nodes of the original diagram and some nodes in the new diagram. If the two graphs are isomorphic, the mapping will be a one-to-one mapping between the graph nodes. The mapping determines which nodes in the original graph guide the placing of which nodes in the new graph.

The 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 practice, 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 5, where the mapping is shown using node labels. In this figure the unlabelled nodes do not participate in the mapping.

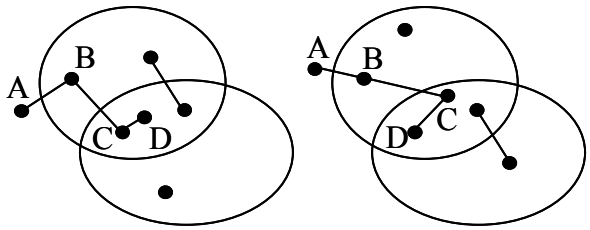


Figure 5. Simple mapping between graph nodes

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. 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. 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. 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 should 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 each zone. The process is iterative, with each iteration including a calculation of force for every node followed by the movement of the nodes according to the force. This movement is capped to prevent very strong forces moving a node out of its zone.

As with the static method, we have repulsive forces acting between each pair of nodes in the zone, and also between nodes and line segments. These forces are given in [14]. In addition, we apply an attractive force to each node that is mapped 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. This encourages each node to become closer to its mapped node, while greatly reducing the chance of the two nodes ending up too far apart. 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 $\frac{kd^2}{c}$.

With three different types of forces acting simultaneously, some care must be taken to choose suitable values for each parameter. 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 this section we show the application of our system to diagrammatic reasoning. Spider diagrams are a subset of *constraint diagrams* [16], 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 a diagram indicates that the set corresponding to that zone is empty. Thus the absence of a zone from a diagram conveys information, and the two spider diagrams in Figure 6 have different semantics.

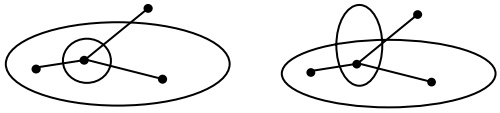


Figure 6. Two equivalent hypergraphs that are different when interpreted as spider diagrams

Each spider in a spider diagram has a *habitat*: the collection of zones that contain nodes of the graph. The spiders assert semantically the existence of elements in the set corresponding to their habitats. 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. The process introduced in this paper for graph drawing can be used for the purposes of spider drawing, but the problems differ slightly in that we have to choose which spider feet are connected with legs. The distinction between drawing graphs and drawing spiders is illustrated in Figure 7 and revisited in section 4.1.

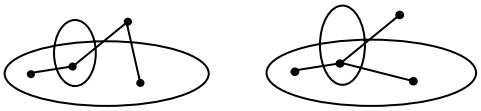


Figure 7. Two equivalent spider diagrams that are different when interpreted as hypergraphs

Once we have given semantics for spider diagrams, we can identify diagram transformations which preserve or weaken the semantic interpretations. Such transformations are called reasoning rules, and a sequence of transformations is a diagrammatic proof: the semantics of the final diagram in a diagrammatic proof are a consequence of the semantics of the initial diagram. Descriptions of reasoning rules can be found in [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.

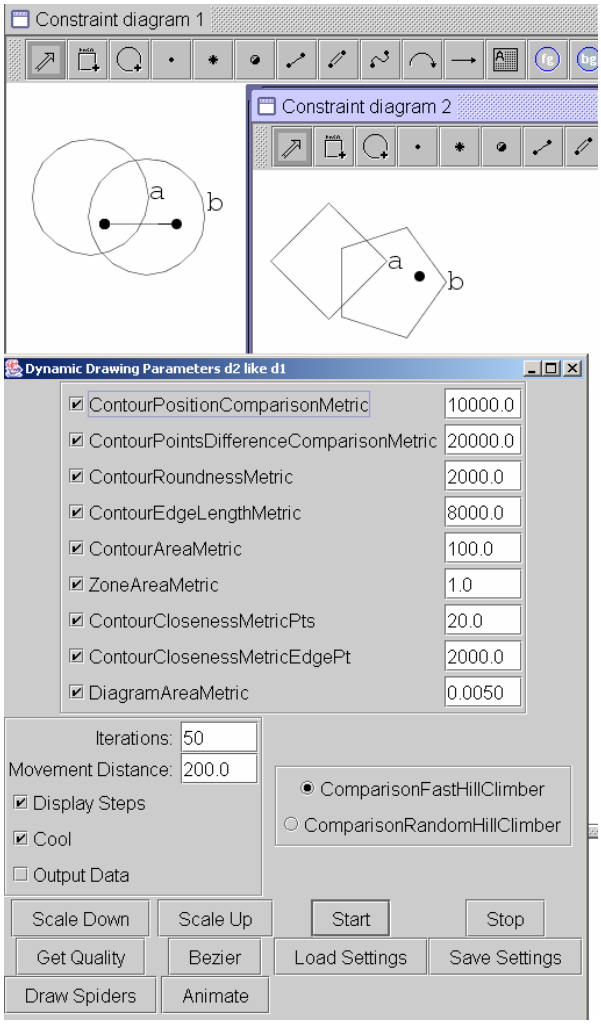


Figure 8. A screen shot of the software

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 intermediate diagrams as abstract descriptions, without layout information. 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.

4.1. Extra Steps Used for This Application

As discussed in [14], 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 applying the static drawing method, 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.

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.

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 now only applies to nodes that are not in the mapping between diagrams.

4.2. Examples

In this section we show the method working on two example proof sequences. All the diagrams in this section are taken from diagrams produced using the application.

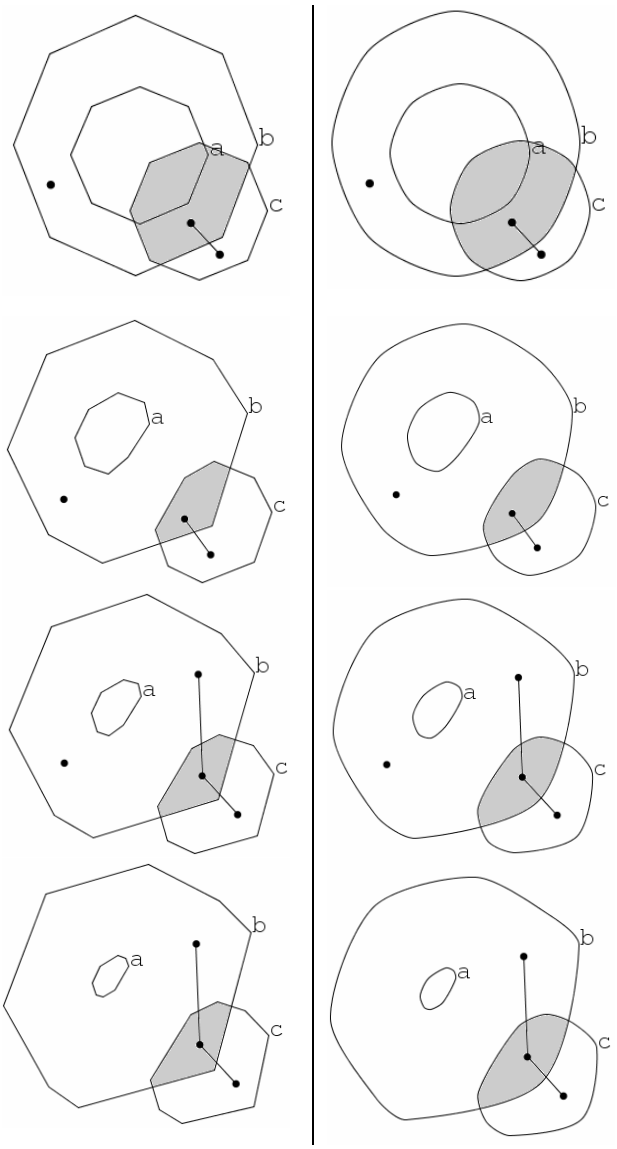


Figure 9. A proof sequence

To make the examples consistent, they have been drawn with the same parameters for Euler diagram multicriteria optimizer and for the graph force algorithm. This inevitably produces a compromise, and better results for individual examples could have been achieved by tuning the numbers. As with most multicriteria 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.

Figure 9 and Figure 10 show proof sequences in which the dynamic drawing method has been applied, drawing each diagram in the context of its predecessor.

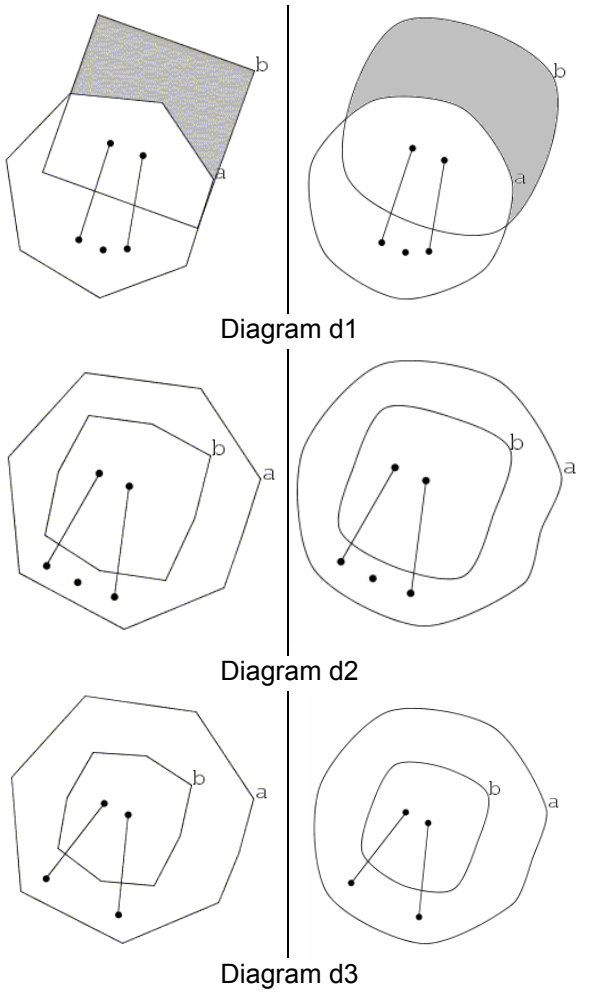


Figure 10. A proof sequence

Both the Euler contours and graphs are maintained in relatively similar positions to the previous diagram, even after structural changes are made. The diagrams on the right are those obtained after using Bezier curves to enhance the smoothness of contours. We regard the layout of these sequences as successful.

Below are the results of applying the metrics to the diagrams in Figure 10. The first column of numbers, d2(d1), shows the results of applying the metrics to d2 in the context of d1. The second column, d3(d2) shows d3 drawn in the context of d2.

These values give the equilibrium position of the hill climber after the optimizing process has finished. The top two metrics are dynamic metrics, dependent on two diagrams, whereas the other metrics are the static metrics, calculated from the single diagram only.

	d2(d1)	d3(d2)
ContourPositionComparison	822.3	5.4
ContourPointsDifferenceComparison	43007.9	312.5
ContourRoundness	62.1	41.9
ContourEdgeLength	22.8	25.9
ContourArea	8.7	14.6
ZoneArea	4.1	4.8
ContourClosenessPts	3018.2	2330.6
ContourClosenessEdgePt	393.4	96.4
DiagramArea	0.022	0.024
TOTAL SCORE	47339.6	2832.1

In the table, the metric values shown have already been scaled so that they are simply added to form the weighted sum. When a metric has a low score (for example, the values of less than 5 for ZoneArea) we can deduce that no amount of diagram manipulation will greatly reduce the metric score further, so we have, in some sense, satisfied that heuristic. On the other hand, a large score (like 822.3 for the ContourPositionComparison) can indicate capacity to reduce this metric score by diagram manipulation. The purpose of including this table is to draw contrasts between the two columns, rather than between the different metrics.

The values reflect the fact that the contour positions in d3 relative to d2 are much better than the positions in d2 relative to d1, as ContourPositionComparison is 5.4 for d3(d2) and 822.3 for d2(d1). We can also see that drawing d3 in the context of d2 has allowed contour points to be moved to similar positions, which were not possible when drawing d2 in the context of d1. This can be seen in the values as ContourPointsDifferenceComparison is 312.5 for d3(d2) and 43007.9 for d2(d1).

5. Further Work

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

Three different forces are applied during the simulation of the force model. More often than not, these complement each other to produce desirable results. However, there are some cases where these forces can be seen to conflict with each other. For example, where one node is on the wrong side of a narrow zone and another node is obstructing its route to its desired location. This kind of problem can be solved by reapplying the initial random layout and

force model. An alternative solution 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.

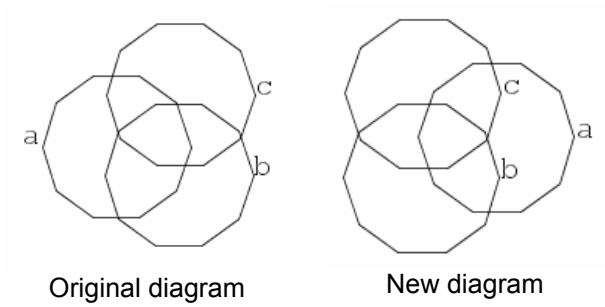


Figure 11. In the new diagram, contour a cannot move to the desired position on the other side of b and c.

The hill climber used as our optimizer can reach local minima, particularly when moving contours that are far away from their desired position, with other contours in the path between current and desired position (see Figure 11). Other more sophisticated optimizers, 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. In other multicriteria systems, occasional random large moves are sometimes made. However, moving a contour a large distance would typically break the structure of the diagram. It would be possible to make larger movements, directed by some heuristic that kept contours close to the other contours with which they intersect. It would still not be possible to guarantee that the structure was maintained, and so multiple attempts might be made, each time testing to see if the structure is correct.

A more sophisticated node mapping than the method described in Section 3.2.1 would achieve a better match between the graphs in the two diagrams. It should be possible to 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.

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 multicriteria 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 has been animated and has been shown to work effectively when 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.

7. Acknowledgements

Thanks to Gem Stapleton at the University of Brighton for helpful comments on drafts of this paper. This work has been 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. Proc. GD2003. LNCS 2912. Springer Verlag. pp. 466-477. 2004.

[4] Consens M.P. and Mendelzon A.O. Hy+: A Hygraph-based Query and Visualization System. 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] Kaufmann M. and Wagner D.. Drawing Graphs: Methods and Models, LNCS 2025. 2001.
[14] Mutton, P.J., Rodgers P.J., and Flower, J.A. Drawing Graphs in Euler Diagrams. To appear in Diagrams 2004. LNAI, Springer-Verlag.
[15] Ruskey F. A Survey of Venn Diagrams. The Electronic Journal of Combinatorics. March 2001.
[16] 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.
[17] 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.