First publ. in: 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, Leganés, 21-25 September 2010 / Christopher D. Hundhausen, Emmanuel Pietriga, Paloma Díaz, Mary Beth Rosson (Eds.), 2010, pp.111-118

DOI: 10.1109/VLHCC.2010.24

Euler Graph Transformations for Euler Diagram Layout

Peter Rodgers
University of Kent
pjr@kent.ac.uk

Gem Stapleton
University of Brighton
g.e.stapleton@bton.ac.uk

John Howse
University of Brighton
john.howse@bton.ac.uk

Leishi Zhang
University of Konstanz
leishi.zhang@uni-konstant.de

***Abstract*— Euler diagrams are frequently used for visualizing information about collections of objects and form an important component of various visual languages. Properties possessed by Euler diagrams correlate with their usability, such as whether the diagram has only simple curves or possesses concurrency. Sometimes, every diagram that represents some given information possesses some undesirable properties, and reducing the number of violations of undesirable properties is beneficial. In this paper we show how to count the number of violations from the reduced Euler graph. We then define various transformations on the Euler graph which can reduce the number of violations of a given property, but sometimes at the expense of increasing the number of violations of another property. These transformations can be used to improve the quality of the drawn diagram, which is important for effective information visualization.**

Keywords-Euler diagrams; Venn diagrams; graph transformations.

I. INTRODUCTION

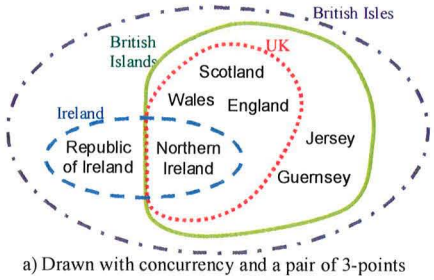
Euler diagrams have been used in a wide variety of applications, including the visualization of statistical data [8], representing non-hierarchical computer systems [2] and in visual semantic web environments [5]. They have also formed the basis of a number of visual languages, including spider diagrams [6], Euler/Venn diagrams [14] and Venn-II diagrams [11]. They generalize Venn diagrams [10] because, although they can contain all possible zones (like Venn diagrams), they do not have to.

Euler diagrams are considered a useful visual tool because they can intuitively show the intersection, containment and disjointness of sets of items, as shown in Figure 1. This figure also serves to motivate the subject matter of this paper. It shows three different ways of displaying the same information, each with different wellformedness properties evident. The diagram in Figure 1a has two curves running together (concurrency) and two tri`ple points (3-points). The diagram in Figure 1b has a self-intersecting curve and one 4-point. Finally, the diagram in Figure 1c has two curves with the same label, and is said to have "duplicated curve labels". In this paper we provide a transformation system for taking a diagram with particular wellformedness properties and converting it to a diagram with alternative properties.

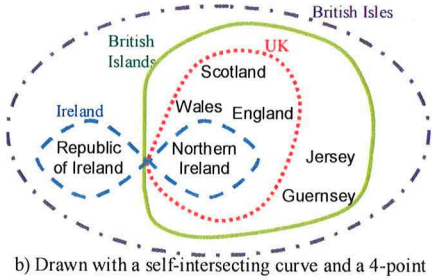
The usability of Euler diagrams is dependent on their appearance. In particular, there are a number of topological features that are considered to have a strong impact on usability. These are termed wellformedness properties, and include: concurrency, n-points, disconnected zones, duplicated curve labels and self-intersecting contours. The terms used here are explained in Section II. It is possible to count the number of occurrences of these properties in a diagram, and one goal of ensuring Euler diagram usability would be to reduce the count.

It is also possible to interchange properties, as shown in Figure 1, where the diagrams show the same information, but have different wellformedness properties. There may be a preferred set of these properties, dependent on the application or user preference. Hence, providing users with a mechanism for modifying a diagram to adjust for their preference would be useful. In addition, generating usable Euler diagrams is non-trivial, and a number of methods for their generation have been proposed. However, typically it not possible to specify the desired wellformedness properties, as the particular properties in the generated diagram are intrinsic to the particular method used [1][9][16]. Hence, after generation, it would be desirable to modify the diagram for user preference.

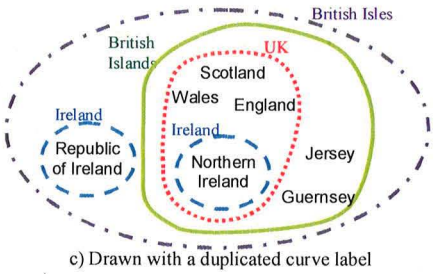
In this paper, we define a number of transformations and explore which Euler diagram properties can be interchanged. The transformations are specified on the Euler graph which is directly derivable from the Euler diagram. After transformations have been performed, the resultant Euler graph can be turned into the appropriate Euler diagram. The alternative approach of applying the transformations directly to the diagram, rather than the Euler graph, is infeasible. This is because the Euler diagram's curves are continuous functions and defining transformations that have the desired effects on such functions is hard. By contrast, graph transformations have been widely studied [3]. Moreover, it is easy to identify the impact of the graph transformations on the drawn Euler diagram. We note one significant difference, in that graph transformations are typically performed on the abstract graph, with no concrete instantiation of the vertex locations or edge routing. Our work relies on performing operations on a plane (drawn) graph, where edge additions are made across particular faces of the graph.



a) Drawn with concurrency and a pair of 3-points



b) Drawn with a self-intersecting curve and a 4-point



c) Drawn with a duplicated curve label

Figure 1. Three Euler diagrams, all showing the same relationship between territories in the British Isles

Previous work in transforming Euler diagrams includes a mechanism to add curves to an Euler diagram using a dual graph of the Euler graph [13]. In addition, transformations on dual graphs that result in curve removal and zone changes have also been developed [4]. The work in both of these previous papers provides a transformation system for completely wellformed diagrams, which possess none of the properties given above. Transformations for the more general, non-wellformed case have been applied to the dual graph [7]. However, these are restricted as they only involve dual graph edge addition or deletions, and are, consequently limited to altering a small number of properties. For instance, the transformation given in Section IV.B cannot be performed in the system of [7] as it would involve changing a node in the dual graph.

In addition, the Euler graph discussed in this paper gives more information about the concrete layout of the Euler diagram compared to the dual graph used in the previous work. For example, dual graphs cannot be used to detect instances of self-concurrency. This paper defines transformations that can be used alter a wider set of wellformedness properties than was previously possible and, therefore, can lead to better Euler diagram usability. Moreover, we show how to count instances of non-wellformedness, and show how these counts are altered by the transformations, further informing usability of diagrams.

In Section II we formally define the concepts of Euler diagram and Euler graph. In Section III we summarize the methods for counting properties of Euler diagrams from the Euler graph. In Section IV we define a set of atomic transformations, which involve the transformation of certain properties. In Section V we show by example how the atomic transformations can be combined into compound transformations to perform further property swaps. In Section VI we give our conclusions and discuss further work.

II. Euler Diagrams and Graphs

We define an **Euler diagram** as a pair $d = (Curve, l)$ where $Curve$ is a finite collection of closed curves with codomain $\mathbb{R}^2$ and $l: Curve \to \mathcal{L}$ is an function that returns curve labels, where $\mathcal{L}$ is a set containing curve labels.

A **minimal region** of an Euler diagram $d = (Curve, l)$, is a connected component of $\mathbb{R}^2 - \bigcup_{c \in Curve} im(c)$, where $im(c)$ is the image of curve c.

In a diagram d, the set of curves with label λ, denoted $Con(\lambda)$ is called a **contour** of d with label λ. A point p is inside $Con(\lambda)$ whenever it is inside a curve with label λ, unless it is inside multiple curves with the label, in which case p is inside $Con(\lambda)$ if it is in a odd number of curves with label λ, otherwise it is exterior to $Con(\lambda)$.

A **zone** is defined to be the set of points interior to a subset of the contours in an Euler diagram and exterior to the others. We observe that, to identify a zone, we only need to know the containing set of contours, since the others can be derived.

We consider a restricted class of Euler diagrams in this paper: we assume that there are finitely many minimal regions and that the multiplicity of any point in $\mathbb{R}^2$ is finite, given any curve. Here, we take the multiplicity of a point, p, given a curve, c, to be the number of times to which p is mapped by c; we denote the multiplicity of p given c by $mul(p, c)$. From a practical perspective, these two restrictions pose no real limitation since we cannot accurately draw Euler diagrams with infinitely many minimal regions or accurately visualise the infinite multiplicity of a point.

Extending the concept of multiplicity, we define the multiplicity of a line segment, e, given curve c to be the minimum multiplicity of any point on e given c, denoted $mul(e, c)$; clearly, $mul(e, c)$ is finite if the multiplicity of each point is finite.

An **abstract description**, D, is a finite subset of $\mathbb{P}\mathcal{L}$ such that $\emptyset \in D$. Elements of D are called **abstract zones**. We abuse the notation for simplification reasons and so write a zone which is interior to contours with labels A, B and C as **ABC**. Similarly, we simplify the notation of an abstract description, so that **{Ø,A,AB}** becomes **Ø A AB.**

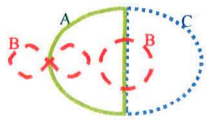
Given an Euler diagram $d = (Curve, l)$ we map d to abstract description $D = ab(d)$, called the **abstraction** of d where $ab(d)$ contains exactly one abstract zone for each zone in d. In particular, given a zone z in d, then $ab(d)$ contains the abstract zone $ab(z) = \{l(c): c \in c(z)\}$, where $c(z)$ is the set of contours that contain z. The abstraction will be used to show how the meaning changes if a transformation alters the zones that are present in a diagram.

Now we define an Euler graph and describe how to convert between Euler graphs and Euler diagrams. Roughly speaking an Euler graph is related to an Euler diagram by having a vertex at each point where two or more curves meet and the edges are the curve segments that connect the vertices. As a special case, curves which do not intersect with other curves are represented by a vertex together with a loop. We are concerned with plane graphs, rather than abstract graphs because we need the edges to be embedded to be able to define the transformations.

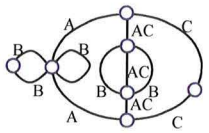
An **unlabelled Euler graph** of an Euler diagram $d = (Curve, l)$ is a plane graph, $G = (V, E)$, with vertex set V and edge set E such that:

1. there are vertices at
    a. any starting point
    b. any turning point, and
2. the embedded edges and vertices have image $\bigcup_{c \in Curve} im(c)$.

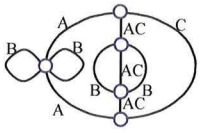
Informally, a starting point of a curve is the point where we start to draw the curve. A turning point is any point where, when drawing the curve, we turn back on the part of the curve just drawn. Note that an unlabelled Euler graph may have additional vertices placed arbitrarily with condition 1, given above, describing a minimal requirement.



a) An Euler diagram, d, with abstraction **Ø A B C AB BC**



b) An Euler graph of d, $EG(d)$



c) A reduced Euler graph of d, $REG(d)$

Figure 2. An Euler diagram, an Euler graph and a reduced Euler graph

Figure 2b shows an Euler graph (with edge labels) of the diagram given in Figure 2a. Vertices must be placed in the graph corresponding to locations where curves intersect in the diagram to ensure planarity. In addition, extra degree two vertices can be present in the graph. However, there are no 'extra' degree two vertices in the graph which could, for example, have arisen from a turning point.

An **edge labelled graph**, is a tuple, $G = (V, E, l_E)$ where (V, E) is a graph with vertex set V and edge set E. The function $l_E: E \to \mathbb{P}(\mathbb{N} \times \mathcal{L})$ labels the edges. If $(n, \lambda) \in l_E(e)$ we say that lambda occurs n times on e.

An **Euler graph** $EG = (V, E, l_E)$, is a plane edge labelled graph where:

1. for each vertex, any given curve label occurs, in total, a even number of times on its incident edges. This means that a closed curve can be formed for every label by traversing edges; and
2. the label of each edge contains at least one occurrence of a curve label.

Let $d = (Curve, l)$ be an Euler diagram. An **Euler graph of d**, $EG(d) = (V, E, l_E)$ is an Euler graph, where (V, E) is an unlabelled Euler graph of d such that each edge, e, is labelled by

$$l_E(e) = \{(mul(e, c), l(c)): c \in Contour\}$$

where $Contour$ is the set of contours in d.

An Euler graph $EG = (V, E, l_E)$ has faces that are the minimal regions of the Euler diagram. We can map each of these faces, f, to an abstract zone using the information in the Euler graph alone. When we convert an Euler diagram to an Euler graph, we can be confident that the abstract description derivable from the Euler graph is the same as the abstract description of the Euler diagram.

A vertex, v, of an edge labelled graph is **redundant** provided it is incident with exactly two distinct edges, each of which have the same labels. For example, the Euler graph in Figure 2b has two redundant vertices: the left-most vertex and the right-most vertex.

A **reduced Euler graph** $REG(d)$ of an Euler diagram d is an edge labelled graph derived from $EG(d)$ by removing redundant vertices, whilst ensuring that $REG(d)$ is homeomorphic to $EG(d)$, until no redundant vertices remain.

Figure 2c shows a reduced Euler graph of the diagram given in Figure 2a. As with an Euler graph, vertices must be placed in the graph corresponding to locations where curves intersect in the diagram. In addition, there are no 'extra' two degree vertices in the graph.

III. Properties of Euler Diagrams

Here we discuss how we enumerate the number of violations of the wellformedness properties using the Euler graph of an Euler diagram. The properties are formally defined for an Euler diagram in [12].

Concurrency. A concurrent line segment is one that is shared by more than one curve. We can count the concurrency in an Euler diagram by considering the edge

labels in the reduced Euler graph. Edges with more than one label correspond to the presence of concurrency. The **edge concurrency count** of an edge e of a reduced Euler graph, REG is:

$$ecc(e, REG) = \left(\sum_{(n,\lambda) \in L_E(e)} n \right) - 1$$

Then REG has **graph concurrency count**:

$$gcc(REG) = \sum_{e \in E(EG)} ecc(e, REG)$$

Given an Euler diagram d and corresponding reduced Euler graph $REG(d)$ the graph concurrency count for $REG(d)$ can be used as a measure of the concurrency in d. For example, the graph concurrency count for the graph in Figure 2c is 3, hence the concurrency count in the diagram in Figure 2a is 3.

***n*-points.** An n-point is a point that is passed through at least n times by curves in the diagram. We can count n-points by looking at vertices of the reduced Euler graph and counting the number of labels on incident edges. Hence, it is possible to find the ***n*-point value** of a vertex, v of a reduced Euler graph, REG using:

$$vnp(n, v, REG) = \max\left\{0, \left(\sum_{\substack{e \text{ is incident} \\ \text{with } v}} \sum_{(m,\lambda) \in L_E(e)} \frac{m}{2} \right) - n \right\}$$

Note we define the summation above to count the value for edge e twice whenever it is a loop. Then REG has **graph *n*-point count**:

$$gnpc(n, REG) = \sum_{v \in V(EG)} vnc(n, v, REG)$$

Given an Euler diagram d and corresponding reduced Euler graph $REG(d)$ the graph n-point count for $REG(d)$ can be used as a measure of the n-point count in d. For example, the 3-point count of the reduced Euler graph in Figure 2c is 3, hence the 3-point count in the Euler diagram in Figure 2a is 3.

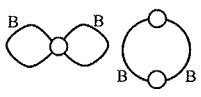


Figure 3. $EG_E(B)$.

Disconnected zones. A zone is disconnected if it comprises more than one minimal region. Since minimal regions correspond to faces in the reduced Euler graph, we can count the number of violations of the disconnected zones property by counting faces. Hence, a reduced Euler graph, REG, has **disconnected zone count:**

$$dzc(REG) = faces(REG) - |Z(d)|$$

where $faces(REG)$ is the number of faces in REG. For example, the number of disconnected zones of the Euler diagrams in Figure 1a and 1b is 0, whereas the disconnected zone count in Figure 2c is 1.

Duplicated curve labels. This is a count of the number of times contours consist of more than one curve. This measure can be derived from the reduced Euler graph by looking at the disconnected subsets of the graph induced by the curve labels. For curve label λ we define $REG_E(\lambda)$ to be the subgraph induced by deleting all edges that do not contain λ and any vertices with degree 0. Then a curve label λ has a non-zero count if $REG_E(\lambda)$ has more than one disconnected component, and the count of a curve label increases as the number of disconnected components increases. We can then sum these for the graph. Hence, a reduced Euler graph, $REG = (V, E)$, has **duplicate curve label count**:

$$dlc(REG) = \sum_{\lambda \in L(G)} \left(comp\left(REG_E(\lambda)\right) - 1 \right)$$

where $comp\left(REG_E(\lambda)\right)$ is the number of disconnected components of $REG_E(\lambda)$. Figure 3 shows $REG_E(B)$ for the reduced Euler graph in Figure 2c. The subgraph derived from the label B is has two disconnected components, and so will contribute 1 to the duplicate curve label count. For example, the duplicate curve label of the reduced Euler graph in Figure 2c is 1 (there is only the disconnected component for label B).

Self-intersecting contours. Finally, we wish to count the number of times a contour self-intersects. If a contour label appears on more than two edges incident with any given vertex then the contour's curve passes through that vertex more than once and, therefore, we have a self-intersection point. Hence, it is possible to find the **self-intersection count** of a vertex, v of a reduced Euler graph $REG = (V, E)$ using:

$$vsic(v, REG) = \sum_{\substack{e \text{ is} \\ \text{incident} \\ \text{with } v}} \left(\left(\sum_{\substack{(n,\lambda) \in L_E(e) \\ \text{and } n>0}} \frac{n}{2} \right) - 1 \right).$$

Again, we count loops twice in the above summation. Extending the definition, REG has **graph self-intersection count**:

$$gsic(REG) = \sum_{v \in V(EG)} vsic(v, REG)$$

Given an Euler diagram d and corresponding reduced Euler graph $REG(d)$ the graph self-intersection count for $REG(d)$ can be used as a measure of the self-intersecting count in d. For example, the graph self-intersection count in Figure 2c is 1, hence the self-intersecting curve count of the diagram in Figure 2a is 1.

IV. Atomic Transformations

In this section we define the atomic transformations that swap between wellformedness properties and which allow more sophisticated compound transformations to be produced. We note that, unlike many graph transformation systems which operate on an abstract graph, the transformations we describe are at the embedded level, where the edge ordering around each vertex is known and, therefore, the faces in the graph can be derived. This ensures that, if the transformations are 'valid', an Euler diagram can be produced from an Euler graph.

We note some general restrictions that are placed on the transformations:

- Transformations are not valid if they do not result in a plane graph. This means that no edge can be added that crosses an existing edge.
- Transformations are not valid if they do not result in an Euler graph. The transformations in this paper always produce an Euler graph.

A. *Edge contraction and expansion*

It is possible to reduce concurrency by contracting an edge, e, that includes at least two contour labels. This can, however, result in a change to the n-point count and the self-intersection count. This can be seen in Figure 4, where moving from the LHS to the RHS has meant that the 4-point count has gone from 0 to 1 (note, however, that the 3-point count remains at 2, because a diagram with a single 4-point has a 3-point count of 2). In addition, because contour **C** is self-intersecting in the RHS, the self-intersection count has increased from 0 to 1. The transformation is symmetric (as are all the transformations defined in this section) and so the reverse operation can conversely add a (possibly concurrent) line segment. Note that in particular cases, for example when the same contour label appears multiple times on the edge, an edge contraction can reduce the n-point count and the self-intersecting count.

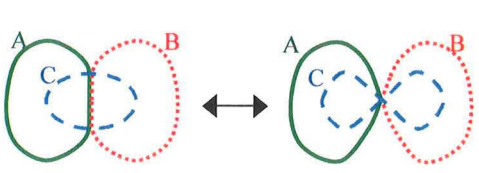


Figure 4. Two embedding of Euler diagram with abstraction **Ø A B AC BC**

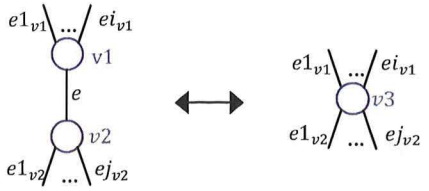


Figure 5. Changing between concurrency and new minimal regions

The transformation that achieves the swap is shown in Figure 5. The LHS shows a line segment of an Euler diagram, represented by an edge labelled e in the Euler graph. This edge can be removed by merging the two vertices, $v1$ and $v2$, into a new, single vertex, $v3$. Both sets of edges emanating from the vertex pair now emanate from the new vertex.

Applying the transformation from LHS to RHS can be performed without restriction. However, there is a constraint when using this transformation from RHS to LHS. The label on e can be anything (as long as it includes at least one contour label) provided, that, for each vertex, v, incident to e, and for all contour labels, λ, in the diagram, the sum of the number of occurrences of λ on the edges incident with e is even. We note that going from the LHS to the RHS does not always mean concurrency is created; it is only created when the number of contour labels for e is more than one. In addition, when going from the RHS to the LHS, the label of the new edge may contain contour labels that were not previously present in the diagram, hence this transformation provides a way of creating new contours. Concurrency is also affected by another technique, as shown in the following subsection.

B. *Edge splitting and merging*

Here we show how concurrency can be reduced at the expense of adding a new minimal region, or vice versa.

When reducing the concurrency count, this operation may either add a new zone (so changing the abstract description), or add a disconnected region to a zone (which maintains the original abstract description). In the example in Figure 6, the LHS Euler diagram has abstract description **Ø A B AB AC**, but the RHS Euler diagram has abstract description **Ø A B C AB AC**, so in this case we added a new zone **C**.

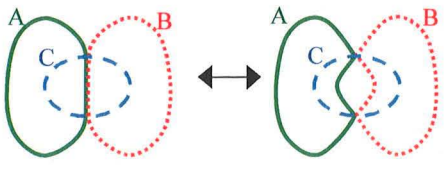


Figure 6. A transformation that swaps between a concurrent line segment and a new minimal region for abstraction **Ø A B AC BC**

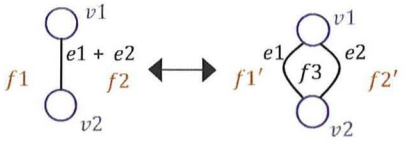


Figure 7. Adding or removing concurrency

We will need to define an operation, +, on edge labels, that merges the two sets of labels: $e1 + e2 = \{(n + m, \lambda): (n, \lambda) \in e1 \text{ and } (m, \lambda) \in e2\}$.

Here, the edge labels $e1$ and $e2$ must each contain at least one contour label, when going from the LHS to the RHS.

Figure 7 shows the transformation definition. If zi is the abstract zone for the zone represented by face fi, then the new abstract zone $z3$ can be derived from $z1$ by, removing contour labels that occur in $z1$ and adding contour labels that do not occur in $z1$. This is because the difference between two adjacent zones can be seen as being the contours that are crossed when going from one to the other. In the special case where the contour label appears an even number of times, the label either appears in both contours or does not appear in both contours. The new abstract zone could equivalently have been calculated from $e2$ and $f2$. The abstract zones derivable from the faces labelled $f1'$ and $f2'$ remain unchanged from the respective faces labelled $f1$ and $f2$.

In addition, no faces apart from the faces labelled $f1$, $f2$, $f1'$, $f2'$ and $f3$ may be affected by the transformation. This prevents newly created edges from being routed in a planar way, but through a face outside the intended scope of the transformation.

C. *Edge deletion and addition*

We can remove curves that run along themselves, at the cost of (possibly) increasing the number of curves used to represent the contour. In the example given in Figure 8, as with figures 5 and 6, both Euler diagrams have abstract description **Ø B C AB AC**.

The transformation definition is shown in Figure 9. This transformation can only be applied if all contour labels occur in edge e an even number of times. The intuition behind this is that, in a Euler diagram, if we cross a contour that is self-concurrent an even number of times, we do not change our relationship (interior or exterior) to that contour.

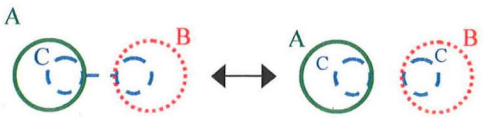


Figure 8. A transformation that swaps between a self-concurrent line segment and duplicate curve labels for abstraction **Ø A B AC BC**

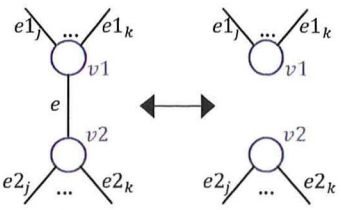


Figure 9. A transformation to add or remove an edge

As with the transformation in Section IV.A, when going from the RHS to the LHS, the edge, e, can include contour labels that are not currently present in the diagram.

V. Combining Transformations

To illustrate the practical application of these transformations, we give two common swaps between wellformedness properties that can be performed using compound transformations.

A. *Interchanging self-intersection, duplicated curve labels and concurrency*

The two wellformedness properties of self-intersecting contours and the number of curves in a contour are, in general, swappable. In addition, it is also possible to swap between some instances of concurrency and self-intersecting contours with an extra step. Here we show how to reduce the concurrency count at the expense of introducing a self-intersecting contour. We can then reduce the self-intersecting count at the expense of increasing the multiple label count.

We illustrate this swap in figure 10 using the British Isles examples given in Figure 1, but with simplified labelling. In this example, going from top to bottom, we first apply an edge contraction (Section IV.A), to Euler Graph 10a to get to Euler Graph 10b, with the corresponding diagram shown on the left, resulting in a swap between concurrency and self-intersection properties. Transforming from Euler Graph 10b to Euler Graph 10c uses three edge expansions (Section IV.A). The final step, transforming from Euler Graph 10c to Euler Graph 10d uses the three edge removals (Section IV.C) to remove the edges labelled **CC**. We note that, as with all the examples in this paper, these transformations can be applied in the opposite order, in this case, going from bottom to top.

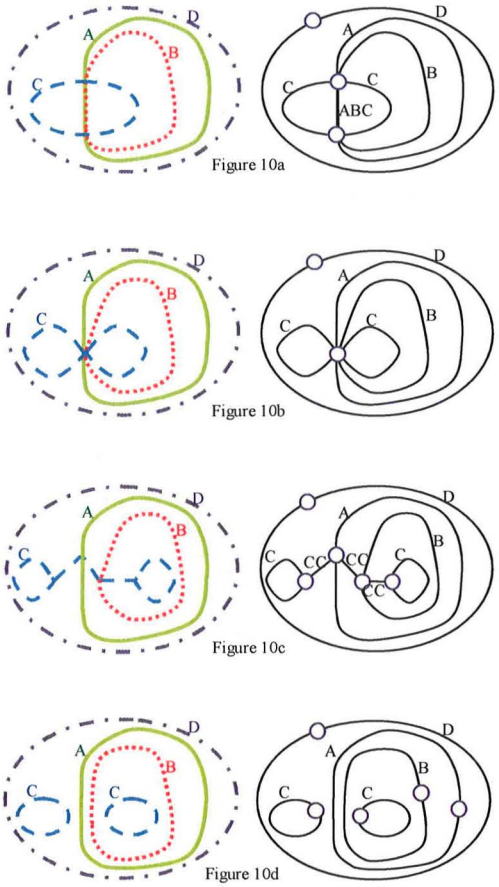


Figure 10. Applying sequences of transformations to produce the diagrams given in Figure 1

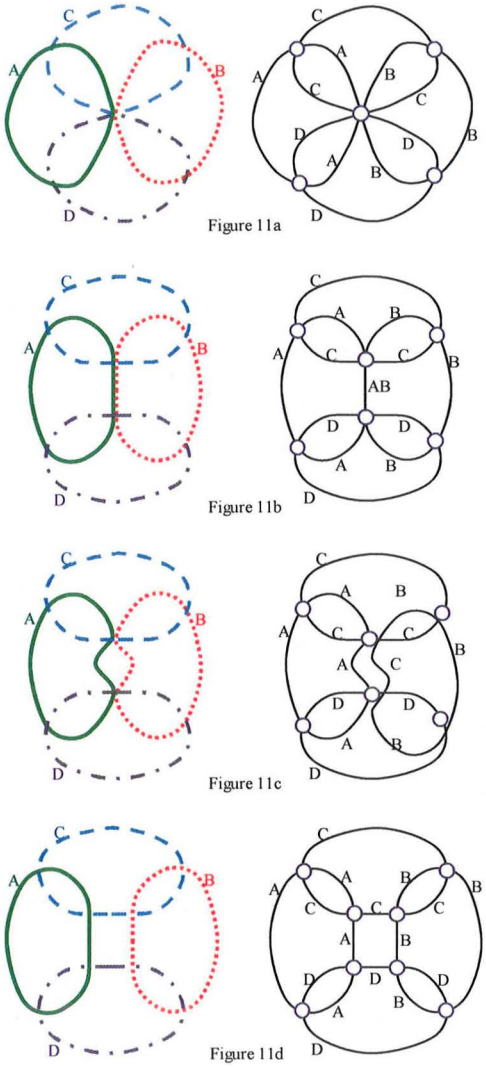


Figure 11. A transformation sequence that interchanges n-points and new minimal regions

Comparing the four diagrams given in Figure 10, we note that Figure 10a has a concurrent line segment, which can make the curves difficult to follow. The concurrent line segment is removed in Figure 10b, but whilst the contour **C** is still connected (and so the items potentially contained in it can still be easily identified), the 4-point makes the contours hard to distinguish when viewed in monochrome, even when different line styles are used. This could then lead to misinterpretation of the diagram. In Figure 10c, contour **C** is still connected and the 4-point has been removed. However, the regions formed contour **C** are more separated and there are more line segments in the diagram. Finally, Figure 10d has reduced complexity (no 3-points and a reduced number of line segments in the final diagram). However, the contour **C** is disconnected, which may cause potential difficulties in understanding which items are contained in **C**.

B. Interchanging n-points and minimal regions

The *n*-points count can be reduced, at the cost increasing the concurrency count or number of minimal regions (or vice versa). In this section, we show a sequence of transformations that achieve this, see Figure 11. The Euler diagram in Figure 11a has a 4-point, where all three curves meet. This point is represented by a vertex with degree 8 in the Euler graph. Removing this triple point requires the application of a number of atomic transformations. Firstly, the Euler graph of Figure 11b shows the result of an edge expansion (Section IV.A) creating a new edge labelled **AB**. The concurrent edge is transformed into a new minimal region to form Euler graph of Figure 11c (Section IV.B). This minimal region is not contained by any contours, so the

abstract description remains the same, but the zone outside all contours is split into two minimal regions. The Euler graph of Figure 11d shows the result of two edge expansions (Section IV.A).

Comparing the diagrams, the 4-point in Figure 11a leads to a symmetric diagram, but might be considered undesirable because of the complexity at the 4-point. Figure 11b reduces the 4-point to two 3-points, at the expense of alternative complexity, the concurrent line segment. Figure 11c, with its split zone and 3-points is unlikely to be preferred over the diagram in Figure 11d, which has the split zone but no 3-points. This lower diagram may have no concurrency or 3-points, but items potentially contained in the outside zone may appear in two distinct minimal regions, meaning that their implicit grouping is not enforced by the visualization.

VI. Conclusion

We have defined transformations that modify Euler diagrams by altering the Euler graph of the diagram. In addition we have defined measures that count the number of violations of properties of Euler diagrams. A benefit to our graph theoretic analysis of Euler diagrams is that we are able to use graph transformations to alter the properties possessed by Euler diagrams. An important consequence of this research is that it lays the foundations for the production of algorithms to automatically produce better diagram layouts which is key if Euler diagrams are to be useful for information visualization and visual languages. Hence, work on examining user preference for various properties, to inform which transformations should be applied, is an important next step in the research in this area.

Future work also includes investigating the completeness of the transformations. Whilst in this paper we have concentrated on transforming diagrams for the particular application of changing wellformedness properties, it would be desirable to ensure that any diagram modification can be made, and it is not clear that it is possible with the atomic transformations given here.

Other further effort is likely to involve implementing the transformations. It is likely that the operation of the transformations can be restricted to only the area of the diagram affected, hence we are optimistic that the system will scale well when large diagrams are altered. One motivation for implementing the transformation system is to enable Euler diagram generation from a library of known good layouts – if we can identify a 'close' existing library diagram to a desired abstract description, then we could transform from the library example to the desired description, and so maintain much of the good layout. In addition, we might wish to draw super diagrams of abstract descriptions which have extra zones from those that are specified in the abstract description. So, given a non-wellformed embedding of an abstract description, by using transformations we could search for a diagram with additional zones that was wellformed.

Acknowledgment

This work has been supported by the EPSRC under grants EP/E010393/1, EP/E011160/1, EP/H012311/1 and EP/H048480/1.

References

[1] S. Chow. Generating and drawing area-proportional Venn and Euler diagrams. Ph.D. thesis, University of Victoria, Canada 2007.

[2] R. DeChiara, U. Erra, V. Scarano. VennFS: A Venn diagram file manager. Proc. IV2003, 120-126. IEEE.

[3] H. Ehrig, G. Engels, H.-J. Kreowski, G. Rozenberg (Editors). Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools. World Scientific. 1999.

[4] A. Fish. Euler Diagram Transformations. In Proc. 8th GTVMG. 85-96. ECEASST 2009.

[5] P. Hayes, T. Eskridge, R. Saavedra, T. Reichherzer, M. Mehrotra, D. Bobrovnikoff. Collaborative knowledge capture in ontologies. In 3rd Int. Conf. on Knowledge Capture. 99-106. 2005.

[6] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil. Spider diagrams: A Diagrammatic Reasoning System. J. Visual Languages and Computing, 12(3):299–324, 2001.

[7] J. Howse, P. Rodgers G. Stapleton. Changing Euler Diagram Properties by Edge Transformation of Euler Dual Graphs. *VL/HCC 2009,* vol. 25, pp. 177-184.

[8] H. Kestler, A. Muller, T. Gress. M. Buchholz. Generalized Venn Diagrams: A New Method for Visualizing Complex Genetic Set Relations. Journal of Bioinformatics, 21(8):1592-1595, 2005.

[9] P. Rodgers, L. Zhang, A. Fish. General Euler Diagram Generation. In Diagrams 2008. 13-27. Springer.

[10] Ruskey, F. A Survey of Venn Diagrams. Electronic Journal of Combinatorics, www.combinatorics.org/ Surveys/ds5/VennEJC.html. 1997.

[11] S.-J. Shin. The Logical Status of Diagrams. Cambridge University Press, 1994.

[12] G. Stapleton, P. Rodgers, J. Howse, J. Taylor. Properties of Euler Diagrams. In *LED 2007.* EASST vol. 7.

[13] G. Stapleton, P. Rodgers, J. Howse, L. Zhang. Inductively Generating Euler Diagrams. IEEE Trans. Visualization and Computer Graphics. In press. doi.ieeecomputersociety.org/10.1109/TVCG.2010.28

[14] N. Swoboda, G. Allwein. Heterogeneous reasoning with Euler/Venn diagrams containing named constants and FOL. Euler Diagrams 2004, ENTCS 134.

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

[16] A. Verroust, M.-L. Viaud. Ensuring the drawability of Euler diagrams for up to eight sets. In Diagrams 2004. LNAI 2980, 128-141. Springer.