ARTICLE IN PRESS

Theoretical Computer Science ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

A graph theoretic approach to general Euler diagram drawing

Gem Stapleton [a,*], John Howse [a], Peter Rodgers [b]

[a] University of Brighton, Brighton, UK
[b] University of Kent, UK

ARTICLE INFO

Article history:
Received 20 November 2008
Received in revised form 16 April 2009
Accepted 2 September 2009
Communicated by G. Ausiello

Keywords:
Euler diagrams
Venn diagrams
Graph drawing
Information visualization

ABSTRACT

Euler diagrams are used in a wide variety of areas for representing information about relationships between collections of objects. Recently, several techniques for automated Euler diagram drawing have been proposed, contributing to the Euler diagram generation problem: given an abstract description, draw an Euler diagram with that description and which possesses certain properties, sometimes called well-formedness conditions. We present the first fully formalized, general framework that permits the embedding of Euler diagrams that possess any collection of the six typically considered well-formedness conditions. Our method first converts the abstract description into a vertex-labelled graph. An Euler diagram can then be formed, essentially by finding a dual graph of such a graph. However, we cannot use an arbitrary plane embedding of the vertex-labelled graph for this purpose. We identify specific embeddings that allow the construction of appropriate duals. From these embeddings, we can also identify precisely which properties the drawn Euler diagram will possess and 'measure' the number of times that each well-formedness condition is broken. We prove that every abstract description can be embedded using our method. Moreover, we identify exactly which (large) class of Euler diagrams can be generated.

1. Introduction

Many diagrams are based on finite collections of (usually simple) closed curves; such a collection of closed curves is called an Euler diagram [1], of which Venn diagrams are examples. To illustrate, the Euler diagram in Fig. 1 contains three closed curves, P, Q and R, which represent collections of objects (sets); it asserts that P and Q are disjoint, and that R may intersect with either P or Q. Areas in which they are used include the visualization of statistical data [2,3], displaying the results of database queries [4] and representing non-hierarchical computer file systems [5]. They have been used in a visual semantic web editing environment [6] and for viewing clusters which contain concepts from multiple ontologies [7]. Another application area is formal object oriented specification [8]. For further application areas, see [9–15]. In all of these areas, automated Euler diagram layout has the potential to bring huge benefits and it is unsurprising that, with the computing power now available, considerable research effort is focused on this topic.

Various methods for automatically generating Euler diagrams have been developed, each concentrating on a particular class of Euler diagrams; for example, see [2–4,16–18]. Ideally, such generation algorithms will produce diagrams with effective layouts in an efficient way. The generation algorithms developed so far produce Euler diagrams that have certain

* Corresponding author. Tel.: +44 0 1273 642410.
E-mail addresses: g.e.stapleton@brighton.ac.uk (G. Stapleton), john.howse@brighton.ac.uk (J. Howse), p.j.rodgers@kent.ac.uk (P. Rodgers).
URLs: http://www.cmis.brighton.ac.uk/research/vmg (G. Stapleton), http://www.cmis.brighton.ac.uk/research/vmg (J. Howse), http://www.cs/kent.ac.uk/people/staff/pjr/ (P. Rodgers).

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.09.005

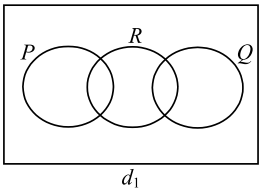

Fig. 1. An Euler diagram.

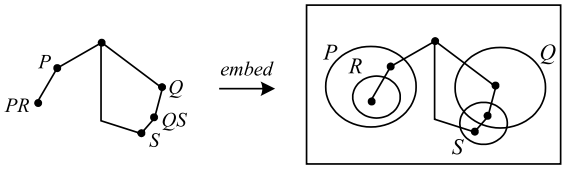

Fig. 2. Generation using a dual graph.

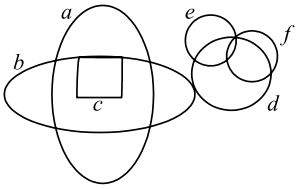

Fig. 3. Properties of Euler diagrams.

sets of properties, sometimes called well-formedness conditions (further discussed below). Each generation method starts with an abstract description of the required diagram and proceeds to seek a layout.

One approach constructs a Venn diagram (a special type of Euler diagram in which all intersections between the curves are present) and removes minimal regions in order to produce the required Euler diagram [19]; this algorithm generates the restricted class of so-called monotonic Euler diagrams which must have a minimal region contained by all of the curves.

Euler diagrams can also be inductively generated, adding one curve at a time [20]. This is an intuitive generation method, since it matches how people typically draw Euler diagrams (at least, based on our experience). This approach to Euler diagram generation can be seen as extending the construction Venn provided in his original paper, where he described how to add curves to Venn diagrams [21]. Edwards also developed an inductive construction for Venn diagrams [22].

Other existing approaches, such as [23,24], construct a so-called dual graph from the abstract description, which is embedded in the plane, and 'wrap' closed curves around the dual graph, as illustrated in Fig. 2; a curve labelled L encloses precisely the vertices that include the label L. Once an appropriate embedding of the dual graph has been found, a layout for each curve in the diagram is determined. It is this graph based generation method that we significantly extend in this paper.

In particular, we develop a very general, formalized framework that allows the generation of an Euler diagram given any abstract description. Moreover, we identify a very large class of Euler diagrams that our method can generate; each abstract description corresponds to many Euler diagrams. In more detail, we define a family of vertex-labelled graphs whose duals give rise to Euler diagrams with the required abstractions. However, we cannot construct arbitrary vertex-labelled graphs for this process and present conditions such that if the graph satisfies those conditions then it will generate an Euler diagram with the correct abstraction. We proceed to use these vertex-labelled graphs to identify properties that the generated Euler diagram will possess. This allows us to choose a vertex-labelled graph from which to generate an Euler diagram based on the properties we wish that diagram to possess.

Two such properties include all curves being simple (simplicity) and each curve having a different label (unique labels). In addition, diagrams may possess curves which never run concurrently (no concurrency), have only connected zones, have no n-points (for some specified integer n) of intersection between curves, and have curves which never 'brush'. To illustrate, in Fig. 3, the curves b and c run concurrently. The zone inside a only is not connected (it consists of two minimal regions). There is a 3-point of intersection between the curves d, e and f, which means this diagram does not possess the no 3-points property (although it does possess the no 4-points property). Finally, the curves b and d intersect at a brushing point; these points are where two curves intersect, but not as part of a concurrent line segment, and do not cross.

Large subsets of these properties are often imposed in generation methods, with the embedded diagrams required to possess the imposed properties. Euler diagrams that possess many of these properties tend to be more visually pleasing or more easily interpreted. However, not every abstract description (formalized below) can be realized under particular collections of these properties (see, for example, [25]). This leads to the necessity of being able to embed Euler diagrams under varying sets of these properties, in part to accommodate user preference when not all of them can hold. A feature of our generation method is the ability to determine which properties hold prior to generation. Moreover, we can select a vertex-labelled graph that allows us to generate a diagram with the required properties, provided such a diagram exists.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

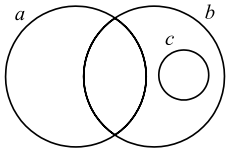

Fig. 4. An Euler diagram.

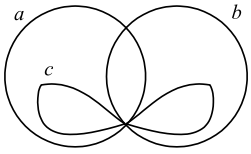

Fig. 5. An Euler diagram with a non-simple curve.

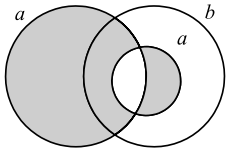

Fig. 6. An Euler diagram with a repeated label.

Section 2 overviews the syntax of Euler diagrams and details properties they may possess and their abstractions. Section 3 introduces a range of graph theoretic concepts that will be required throughout the paper. Section 4 describes how to convert the so-called appropriate embeddings of vertex-labelled graphs into Euler diagrams. We observe various properties possessed by the graph that establish properties that will be possessed by the yet to be generated diagram; this is done in Section 5. We sketch a naive algorithm for generating a family of graphs in Section 6, from which we can generate Euler diagrams with specified properties. Finally, Section 7 presents a prototype implementation of our work, which constructs an Euler diagram given a vertex-labelled graph. Some output from the software is included.

2. Euler diagrams

An Euler diagram is a collection of closed curves, each of which has a label which we assume is drawn from a fixed set $\mathcal{L}$. There are many definitions of Euler diagrams, each asserting *well-formedness conditions* that the diagrams must satisfy, such as the curves should be simple, for example [2–4,23,26]; we give a very general definition.

Definition 1. An **Euler diagram**, d, is a pair, $(Curve, l)$ where

(1) $Curve$ is a finite collection of closed curves each with codomain $\mathbb{R}^2$,
(2) l: $Curve \to \mathcal{L}$ is a function that returns the label of each curve.

Fig. 4 shows an Euler diagram with three curves labelled a, b and c. It is easy to determine the interior or exterior of the (simple) curves in this diagram. However, some Euler diagrams contain non-simple curves. A non-simple curve is one which self-intersects. For example the curve c in Fig. 5 is non-simple. This diagram indicates that c is a subset of the symmetric difference of a and b. We now define the interior points of such curves, by appealing to winding numbers. Thus, we assume that the curves in Euler diagrams allow the computation of winding numbers.

Definition 2. Let c be a closed curve and let p be a point in $\mathbb{R}^2 - im(c)$. The point p is **interior** to c if the winding number of c with respect to p, denoted $wind(c, p)$, is odd, with the set of all such points denoted $int(c)$. All points in $\mathbb{R}^2$ that are not interior to c are **exterior** to c, with the set of all such points denoted $ext(c)$.

The labelling function l is need not be injective and it is necessary to consider the set of all curves that have the same label; we call this a contour. Curves with the same label represent the same set.

Definition 3. Let $d = (Curve, l)$ be an Euler diagram and let $Con(L)$ be the set of all curves in d with the label L. The set $Con(L)$ is a **contour** of d with label L. A point p is **inside** $Con(L)$ whenever the number of curves in $Con(L)$ that p is inside is odd. The set of interior points is denoted $int(Con(L))$. All points in $\mathbb{R}^2$ that are not interior to $Con(L)$ are **exterior** to $Con(L)$, the set of which is denoted $ext(Con(L))$.

The Euler diagram in Fig. 6 has two contours. The label a is associated with two of its curves. The shaded regions of the diagram indicate the interior of the contour with label a.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

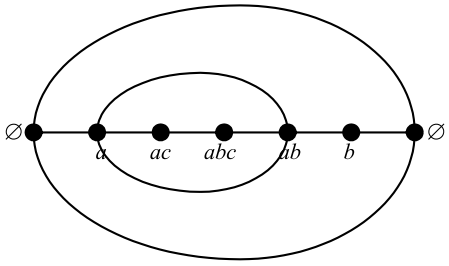

Fig. 7. A connected planar graph with vertices labelled by zones.

Definition 4. Let $d = (Curve, l)$ be an Euler diagram and let $\mathcal{CON} \subseteq \{Con(L) : L \in im(l)\}$. If the set

$$z = \bigcap_{Con(L') \in \mathcal{CON}} int(Con(L')) \cap \bigcap_{Con(L') \in \{Con(L) : L \in im(l)\} - \mathcal{CON}} ext(Con(L'))$$

is non-empty then z is a **zone** of d, with the set of such zones denoted $Z(d)$.

The diagram in Fig. 4 has five zones, including that inside both a and b but outside c. The diagram in Fig. 6 has four zones; the zone inside b but outside a consists of two disjoint regions, as does the zone inside both a and b.

Our attention now turns to descriptions of diagrams. We can provide an abstract description of an Euler diagram by considering its zones, as has been well documented in the literature. A zone can be represented by the labels of the curves that contain it.

Definition 5. Elements of $\mathcal{Z} = \mathbb{P}\mathcal{L}$ are called **abstract zones** (or, simply, zones). An **abstract description**, D, is a set of abstract zones, $D \subseteq \mathcal{Z}$ such that $\emptyset \in D$.

Definition 6. Given an Euler diagram $d = (Curve, l)$, we map d to $ab(d) = D$, called the **abstract description** of d, where D contains exactly one abstract zone for each zone in d; in particular, given a zone, z, in d, the set D contains the abstract zone

$$ab(z) = \{l(c) : c \in C(z)\}$$

where $C(z)$ is the set of curves in d that contain z.

In Fig. 4, the Euler diagram has abstract description

$$\{\emptyset, \{a\}, \{b\}, \{a, b\}, \{b, c\}\}.$$

However, for ease of readability, we will abuse the notation by representing this abstraction as $\{\emptyset, a, b, ab, bc\}$.

In its simplest form the *Euler diagram generation problem* can be summarized as: given an abstract description, D, find an Euler diagram with abstract description D. Research efforts in this area have focused on a restricted notion of the generation problem: given D, find an Euler diagram with abstraction D that possesses certain specified properties. Ideally, we would have a generation algorithm that produced an Euler diagram when an arbitrary set of properties (chosen from the six described in the introduction) is specified. The framework presented in this paper provides a basis for the development of sophisticated generation algorithms that achieve this. Moreover, if an abstract description can be visualized as an Euler diagram with a certain set of these six properties then our method is able to generate such a diagram. However, without heuristics to guide the search for an appropriate graph this process will have exponential time complexity. We are able to adapt the graph we use for layout, by adding or deleting edges or vertices, in order to change the properties that the embedded Euler diagram will possess.

3. Graph theory concepts

We will show that an Euler diagram can be generated from *any* connected planar graph whose vertices are labelled with the zones from an abstract description. For example, consider the abstract description $D = \{\emptyset, a, b, ab, ac, abc\}$. Fig. 7 shows a connected planar graph whose vertices are labelled with the zones of D. Each zone of D must occur as a label and a zone can label multiple vertices. We draw closed curves around the vertices containing common labels, shown on the left in Fig. 8. Throwing away the graph we obtain the required Euler diagram, shown on the right in Fig. 8. We notice that the diagram obtained can be constructed from a dual of the graph, ignoring the infinite face, as shown in Fig. 9. Our strategy for generating an Euler diagram d from an abstract description D is to construct a dual of a connected planar graph, G, whose vertices are labelled with the abstract zones of D. However, given an arbitrary G, not every dual gives rise to a diagram with the required abstraction, making the generation task more challenging than merely constructing a dual graph.

Indeed, if we want to find an Euler diagram with specified properties then we must take into account various factors. In particular, given an abstract description, D, if we want to find an embedding of D with certain properties then this amounts to finding a graph with related properties. In this paper, we take the important first step towards addressing this challenge by identifying properties of the embedded graph that correspond exactly to properties that the embedded Euler diagram will posses. We give formal definitions of the graphs we require.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

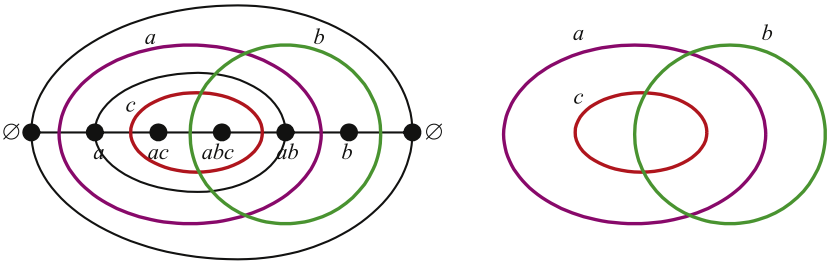

Fig. 8. Contours containing vertices and the resulting Euler diagram.

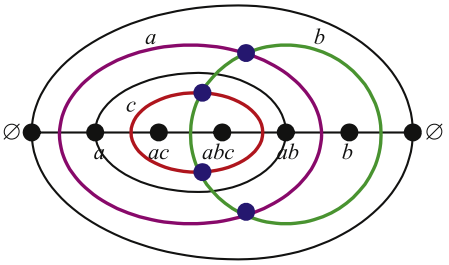

Fig. 9. The dual of the graph.

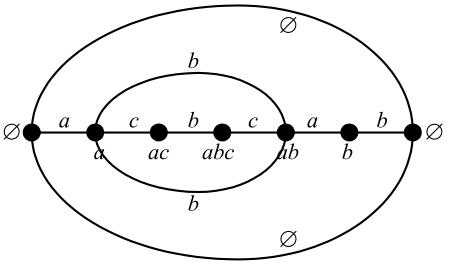

Fig. 10. A vertex-labelled graph.

Definition 7. A **vertex-labelled graph**, $G = (V, E, l_V, l_E)$, is such that

(1) (V, E) is a graph with vertex set V and edge set E,
(2) each vertex, v, in V is labelled by a zone, that is $l_V : V \to \mathcal{Z}$, and
(3) the vertex labelling induces an edge labelling, $l_E : E \to \mathcal{Z}$, defined by

$$l_E(e) = (l_V(v_1) - l_V(v_2)) \cup (l_V(v_2) - l_V(v_1))$$

where v_1 and v_2 are the vertices incident with e.

Fig. 10 shows a vertex-labelled graph. The label on an edge is the symmetric difference of the labels of its incident vertices. The induced edge labelling is redundant, and we may frequently exclude it in examples. However, it will be useful for obtaining the labels of the edges in dual graphs, as will become clear below.

Definition 8. An **edge-labelled graph**, $G = (V, E, l_E)$, is such that

(1) (V, E) is a graph with vertex set V and edge set E,
(2) each edge, e, in E is labelled by a zone, that is $l_E : E \to \mathcal{Z}$.

We are labelling edges with a set of labels from $\mathbb{P}\mathcal{L}$ (that is a zone, but not necessarily a zone that occurs in the abstract description we are considering or as a vertex label on a vertex-labelled graph) and, for each $L \in \mathcal{L}$ we say that L is a label of e if the label L is in the set $l_E(e)$. We now define further concepts that we will require throughout the paper. In particular, we need access to certain subgraphs, edge sequences around faces, and the number of components of which a graph consists.

Definition 9. Let G be a graph. Then $comp(G)$ denotes the number of components of G.

Definition 10. Let $G = (V, E, l_V, l_E)$ be a planar graph with an embedding, $\hat{G}$. Let f be a face of $\hat{G}$. Then an **edge sequence around f** is a minimal sequence of edges, $ES(f) = (e_1, \ldots, e_n)$ such that

(1) each edge in $ES(f)$ bounds f,
(2) each edge that bounds f occurs in $ES(f)$, and
(3) $ES(f)$ is a closed walk.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

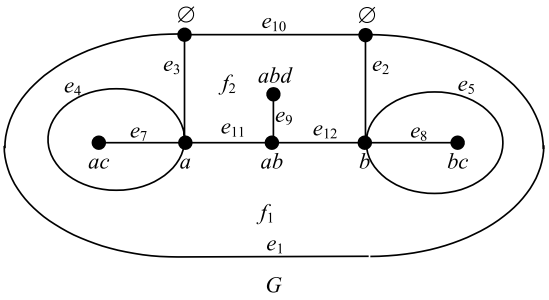

Fig. 11. Face cycles.

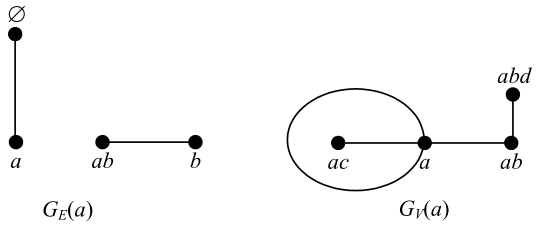

Fig. 12. *L*-edge and *L*-vertex subgraphs.

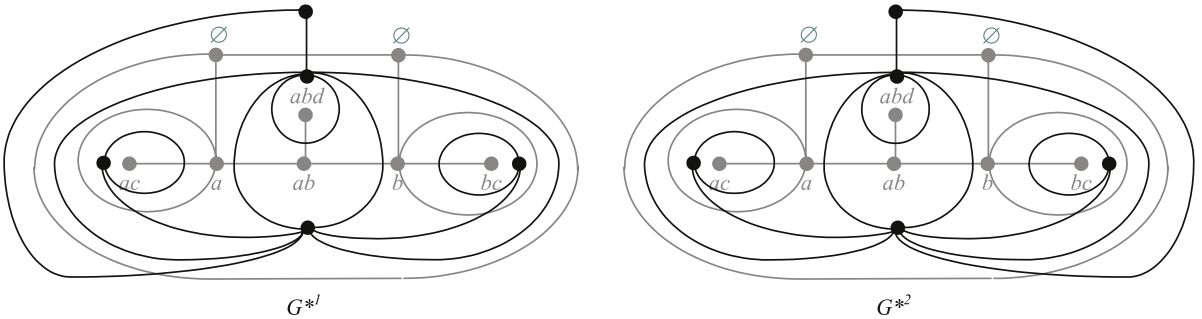

Fig. 13. Two distinct dual graphs.

To illustrate, f_1 in Fig. 11 has edge sequences $(e_1, e_2, e_5, e_{12}, e_{11}, e_4, e_3)$ and $(e_2, e_5, e_{12}, e_{11}, e_4, e_3, e_1)$ among others (note the 'labels' used on the edges are not edge labels, merely names for the edges so that we can identify them). The closed walk $(e_1, e_2, e_5, e_{12}, e_{11}, e_3)$ is not an edge sequence around f_1 since it does not contain e_4. The face f_2 has edge sequence $(e_{10}, e_3, e_{11}, e_9, e_9, e_{12}, e_2)$; notice that the bridge e_9 occurs twice in the walk.

Definition 11. Let $G = (V, E, l_E)$ be a vertex-labelled graph and let L be a label in $\mathcal{L}$. The subgraph of G obtained by deleting all edges whose labels do not contain L (along with any isolated vertices), denoted $G_E(L)$, is the ***L*-edge subgraph** of G given L. The subgraph of G obtained by deleting all vertices whose labels do not contain L, denoted $G_V(L)$, is the ***L*-vertex subgraph** of G given L. Let az be an abstract zone. The subgraph of G obtained by deleting all vertices whose labels are not az, denoted $G_V(az)$, is the ***az*-vertex subgraph** of G given az.

Note that the above definition of $G_E(L)$ extends to edge-labelled graphs in the obvious manner. To illustrate these concepts, the graph G in Fig. 11 has $G_E(a)$ and $G_V(a)$ as illustrated in Fig. 12. Finally, since our embedding method relies on constructing a specific dual graph, we define an edge-labelled dual.

Definition 12. Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph with a plane embedding $\hat{G}$. Let $\hat{G}^* = (V^*, E^*, l_{E^*})$ be an edge-labelled geometric dual of $\hat{G}$ such that for each edge, $e^* \in E^*$, $l_{E^*}(e^*) = l_E(e)$ where e^* crosses e. The (plane, embedded) graph $\hat{G}^*$ is called an **edge-labelled dual** of $\hat{G}$.

Fig. 13 shows two distinct geometric duals of G in Fig. 11. The duals are different, up to isotopy of $\mathbb{R}^2$ less the images of G's vertices. For example, the vertex in G next to the infinite face of G^{*1} is distinct from that next to the infinite face of G^{*2}. This is an important observation since we use a dual, G^*, of a graph, G, to embed an Euler diagram. If that Euler diagram is to have the required abstraction, the vertices of G have to be enclosed by the correct curves (further examples will be given below). This means that not every dual need give rise to an Euler diagram with the required abstraction.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

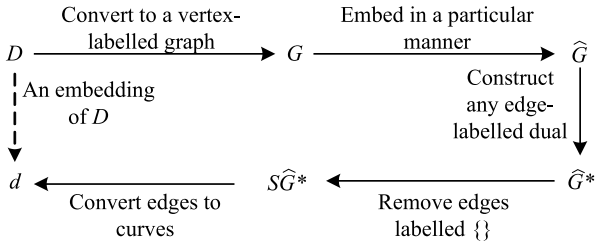

Fig. 14. The generation process.

4. Embedding Euler diagrams

Our attention now turns to the challenge of converting a graph into an Euler diagram. The generation process is outlined in Fig. 14. We start with an abstract description, D, that we want to embed. We then turn D into a vertex-labelled graph, G, such that every abstract zone in D appears as a vertex label in G *and* every vertex label is an abstract zone in D. We proceed to embed G, giving $\hat{G}$, in such a manner that G has only vertices and edges labelled $\emptyset$ next to the infinite face; if this is not possible then we can add edges to G so that it becomes possible (note that we never need to add vertices, but we can choose to if we wish).

At the next stage, we construct an edge-labelled dual, $\hat{G}^*$, of $\hat{G}$. The fact that every edge next to the infinite face is labelled $\emptyset$ ensures that $\hat{G}^*$ will give rise to a diagram with abstraction D (explained and justified below). The labels on the edges of $\hat{G}^*$ indicate that curves with those labels run along that edge. Hence, edges labelled $\emptyset$ will not be traversed by any curves and are deleted, to give a graph $S\hat{G}^*$. At this point, $S\hat{G}^*$ has the same image as the Euler diagram we will create. The final step is to turn this graph into the Euler diagram (i.e. a set of closed curves together with a labelling function). The method we develop to produce a set of curves ensures that they have the correct containment properties. That is, a vertex labelled az is embedded in a zone with abstraction az.

When producing an embedding of a plane graph, typically the embedded edges, e, are injective functions with codomain $\mathbb{R}^2$, say $\hat{e}\colon [x, y] \to \mathbb{R}^2$. Moreover, $\hat{e}(x)$ and $\hat{e}(y)$ are the points in $\mathbb{R}^2$ where the vertices incident with e are embedded. We assume that this is the case in all that follows.

We now develop a series of results that are required (a) to show that we can covert the dual graph into curves, and (b) that the result is an Euler diagram with the required abstraction. We note that the following property of vertex-labelled graphs follows trivially from the manner in which edges are labelled.

Property 1. *Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph. Let $p = (e_1, \ldots, e_n)$ be a path in G with associated vertex sequence $(v_0, \ldots, v_n)$. Let $L \in \mathcal{L}$. If*

(1) *$L \in l_V(v_0)$ and $L \in l_V(v_n)$, or*
(2) *$L \notin l_V(v_0)$ and $L \notin l_V(v_n)$*

then the maximal subsequence of p that contains only edges, e, for which $L \in l_E(e)$ has even length.

We use this property in the proof of the next result, which asserts that components of $\hat{G}^*_E(L)$ (the subgraph of $\hat{G}^*$ containing exactly the edges with L in their label) are Eulerian. We require this to be the case in order to convert $\hat{G}^*_E(L)$ to curves: a closed curve can be formed by traversing an Eulerian cycle, for instance.

Lemma 1. *Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph with a plane embedding $\hat{G}$. Let $L \in \bigcup_{e \in E} l_E(e)$ (i.e. the L label that appears on an edge of G). Then each component of the graph $\hat{G}^*_E(L)$ is Eulerian.*

Proof. The task is to show that each vertex, v, in $\hat{G}^*_E(L)$ has even degree. By construction, v is embedded in a face, f, of $\hat{G}$. From Property 1, the maximal subsequence, S, of edge sequence $ES(f)$ that consists of those edges that contain the label L has even length (since $v_0 = v_n$ in this case). Any edge, e, in S that occurs twice is a bridge and, therefore, gives rise to a loop in the dual graph $\hat{G}^*$. Thus, such an edge contributes two to the degree of v whenever L is in $l_E(e)$. No edge occurs more than twice. Denote the subsequence of S that does not contain any bridges by S_{nb}. Then S_{nb} also has even length and each edge in S_{nb} contributes one to the degree of v. Therefore v has even degree and each component of $\hat{G}^*_E(L)$ is Eulerian. □

As a simple illustration of the process of converting an Eulerian cycle to a curve, consider the graph, $\hat{G}$, in Fig. 15, together with its dual $\hat{G}^*$. The graph $\hat{G}^*(a)$ is shown and a curve labelled a can, intuitively, be formed by traversing the edges of $\hat{G}^*(a)$, following an Eulerian cycle. In the following definition, $\hat{e}_i|_{[i-1,i)}$ denotes the function $\hat{e}_i$ with the domain is restricted to $[i-1, i)$.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

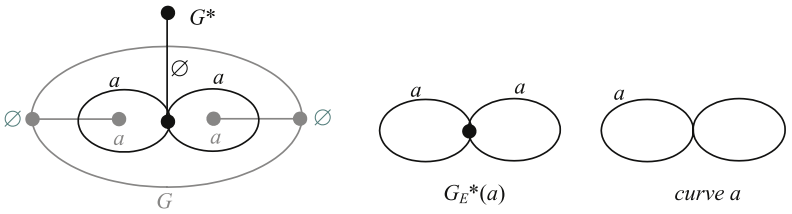

Fig. 15. Converting cycles to curves.

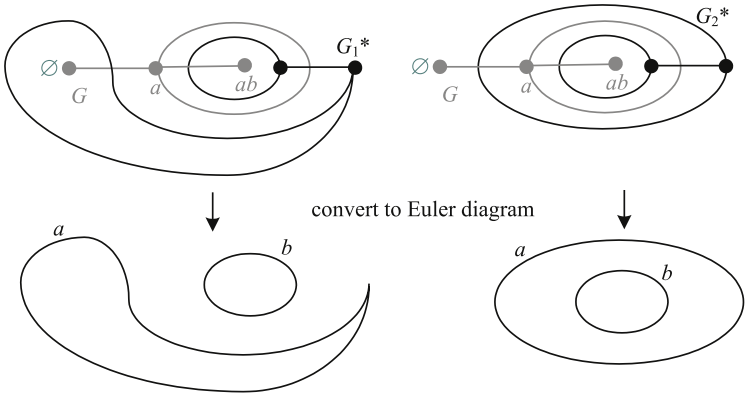

Fig. 16. Finding appropriate embeddings.

Definition 13. Let $\hat{G}$ be a plane (and, therefore, embedded), Eulerian graph and let $C = (e_1, \ldots, e_n)$ be an Eulerian cycle in G. For each edge, e_i, we denote the injective function that embeds e_i in $\mathbb{R}^2$ by $\hat{e}_i$ and assume that the domain of $\hat{e}_i$ is $[i-1, i]$, that is $\hat{e}_i\colon [i-1, i] \to \mathbb{R}^2$. Then the closed curve **associated** with $\hat{G}$ given C, is defined to be

$$\hat{G}_C = \bigcup_{1 \le i \le n} \hat{e}_i|_{[i-1,i)} \cup \hat{e}_n(n).$$

So $\hat{G}_C\colon [0, n] \to \mathbb{R}^2$ is a closed curve with a finite number of self-intersection points. These self-intersection points coincide precisely with the vertices in the cycle that have degree greater than two. We are now in a position to state how to convert a vertex-labelled graph into an Euler diagram.

Definition 14. Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph with a plane embedding, $\hat{G}$. An Euler diagram, $d = (Curve, l)$, **generated** from $\hat{G}$, given a dual $\hat{G}^*$, is such that for each label $L \in \bigcup_{e \in E} l_E(e)$ and for each component EC of $\hat{G}^*_E(L)$ the closed curve EC_C is in $Curve$ given some Eulerian cycle, C, in EC; the label of C is L. No other curves are in $Curve$.

Later, we widen the range of Euler diagrams that can be generated by considering decompositions of Eulerian cycles into sets of cycles. This corresponds directly to decomposing curves into sets of curves.

4.1. The correctness of the generation method

As stated above, to ensure we generate an Euler diagram with the required abstraction, we must impose certain constraints on our vertex-labelled graphs. In particular, when we embed them, we require the infinite face to be next to only edges that are labelled $\emptyset$.

Suppose, for example, we want to embed the abstract description $D = \{\emptyset, a, ab\}$. A vertex-labelled graph, G, with suitable vertex labels can be seen in Fig. 16; this embedding does not have only edges labelled $\emptyset$ embedded next to the infinite face of G. The dual graph G_1^* of G gives rise to an Euler diagram with abstract description $\{\emptyset, a, b\}$ which is not as required. However, G_2^* gives rise to a diagram with the correct abstraction. We observe that there are choices about how the edges of dual graphs are embedded that impact the containment properties of the resulting curves. However, there are only choices (up to isotopy) for edges of the dual that are embedded in either the infinite face or other faces that are not simply connected (i.e. they contain holes).

In the case of a dual vertex, v, embedded in the infinite face, f, of the original graph, if we know that all of the edges in G next to f are labelled $\emptyset$ then every edge incident with v is also labelled $\emptyset$. Such edges do not form part of any curve in the generated Euler diagram. Therefore, any choice made when embedding these edges does not matter. We also stipulate that our embedded graphs have only vertices labelled $\emptyset$ next to the infinite face. As a consequence of these constraints on G, vertices will be in zones whose abstraction matches the label of the vertex.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

Definition 15. Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph with an embedding, $\hat{G}$. If $\hat{G}$ is plane, every edge, $e \in E$, embedded next to the infinite face of $\hat{G}$ has label $l_E(e) = \emptyset$ and is incident with a vertex, v, whose label is $l_V(v) = \emptyset$, then $\hat{G}$ is called an **appropriate** embedding of G.

The case where a face is not simply connected is easily overcome: we require G to be connected. Any connected graph has only simply connected faces. Given a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$, we can establish that $\hat{G}$ generates an Euler diagram with the required abstraction. Theorem 1 is a prerequisite to establishing this result.

Theorem 1. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. Let $L \in im(l)$ and let $v \in V$. Then $L \in l_V(v)$ if and only if v is inside the contour of d with label L.*

Proof. By definition, $Con(L)$ arises from the graph $\hat{G}^*_E(L)$ which has the property that all of its components are Eulerian graphs. From this property, it follows that $\hat{G}^*_E(L)$ is two face-colourable. Choose such a colouring of $\hat{G}^*_E(L)$. Let v_1 and v_2 be vertices of $\hat{G}$ that are embedded in faces of $\hat{G}^*_E(L)$ that have the same colour. Since $\hat{G}$ is connected, there exists a path, P, from v_1 to v_2. Now, every edge, e, in P that crosses an edge in $\hat{G}^*_E(L)$ has L in its label, that is $L \in l_E(e)$. Since v_1 and v_2 are in faces with the same colour, there are an even number of such edges in P. Moreover, each edge, e, in P that contains L in its label has exactly one incident vertex with L in its label (by definition). From this it follows that v_1 and v_2 either both have L in their label or neither have L in their label.

Now consider the infinite face of $\hat{G}^*_E(L)$. The vertex, v^*, of $\hat{G}^*$ embedded in the infinite face of $\hat{G}$ is incident only with edges that have label $\emptyset$, since every edge, $e \in E(\hat{G})$, embedded next to the infinite face of $\hat{G}$ has label $l_E(e) = \emptyset$. Therefore the edges incident with v^* are not in $\hat{G}^*_E(L)$ and, therefore, v^* is not a vertex of $\hat{G}^*_E(L)$. From this, it follows that some vertex, v, of $\hat{G}$ incident with an edge embedded next to the infinite face of $\hat{G}$ is embedded in the infinite face of $\hat{G}^*_E(L)$. This vertex has label $l_V(v) = \emptyset$, since $\hat{G}$ is an appropriate embedding. Therefore, any vertex in a face of $\hat{G}^*_E(L)$ coloured the same as the infinite face of $\hat{G}^*_E(L)$ does not contain L in its label. Those coloured differently from the infinite face of $\hat{G}^*_E(L)$ all contain L in their labels.

A point is inside $Con(L)$ precisely when it is inside a face of $\hat{G}^*_E(L)$ that is coloured differently from the infinite face of $\hat{G}^*_E(L)$ in some two colouring. Hence, $L \in l_V(v)$ if and only if v is inside the contour of d with label L. □

Corollary 1. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. Let $v \in V$. Then $l_V(v) = ab(z)$ where z is the zone of d in which v is embedded.*

Therefore, we generate diagrams with the required abstractions.

Theorem 2. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. Then $ab(d) = im(L_V)$.*

4.2. Curve decomposition

Given an embedded Euler diagram, we can find alternative Euler diagrams that have the same drawn image and the same abstraction. Here, we show how to use the techniques of the previous section to enlarge the class of Euler diagrams that can be embedded using our method. We start by observing that an Eulerian graph can have its edges partitioned into sets, each of which is essentially a cycle in the graph.

Definition 16. Let G be an Eulerian graph and let $\mathcal{C}$ be a set of cycles in G. If every edge in G is in at exactly one cycle in $\mathcal{C}$ then $\mathcal{C}$ is called a **decomposition** of G. If every cycle in a decomposition $\mathcal{C}$ is simple[1] then we say that $\mathcal{C}$ is a **simple decomposition**. Let c be the closed curve associated with G. Then a **decomposition** of c is a set of curves, $dec(c)$, where each cycle in $\mathcal{C}$ is associated with exactly one curve in $dec(C)$. Similarly, $dec(c)$ is a **simple decomposition** of c whenever $\mathcal{C}$ is a simple decomposition.

Definition 17. Let $G = (V, E, l_V, l_E)$ be a planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. For each curve, $c \in Curve$, let $dec(c)$ be a decomposition of c. Then a **decomposition-transformation** of d is a diagram, $d' = (Curve', l')$, where

$$Curve' = \bigcup_{c \in Curve} dec(c)$$

and $l'(c') = l(c)$ where c' is in $dec(c)$.

[1] A simple cycle is one that does not pass through any vertex more than once.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

Theorem 3. *Let d be an Euler diagram with a decomposition-transformation d'. Then $ab(d) = ab(d')$. That is, decomposing curves does not change the abstraction.*

The results above allow us to generate a wider class of Euler diagrams by decomposing their curves. In particular, we point out every abstraction can be drawn using only simple curves, justified below.

Lemma 2. *Let G be an Eulerian graph and let $\mathcal{C}$ be a decomposition of G. If $\mathcal{C}$ is a simple decomposition then all curves in a simple decomposition of c, the curve associated with G, are simple.*

Lemma 3. *Let G be an Eulerian graph. Then G has a simple decomposition.*

Proof. We provide a constructive method to produce a simple decomposition. Clearly, one can always find a simple cycle in an Eulerian graph. Given a simple cycle, C_1, in G, the graph $G - C_1$ has only Eulerian components (since deleting edges in C_1 leaves all vertices with even degree). Continuing in this manner, we can keep removing simple cycles, C_i, from G until no edges remain. Therefore there is some integer, n, such that removing n simple cycles results in a graph with no edges. The set of removed simple cycles, $\mathcal{C} = \{C_1, \ldots, C_n\}$, is a simple decomposition of G. □

Corollary 2. *Let G be an Eulerian graph. Then the curve associated with G has a simple decomposition.*

Theorem 4. *Every Euler diagram generated using our method can be transformed into an Euler diagram containing only simple curves.*

4.3. Sufficiency and classification

Trivially, it can be shown that for every abstract description there is a connected, planar vertex-labelled graph that yields an Euler diagram with that abstract description. For instance, taking $D = \{\emptyset, a, b, bc\}$, define $V = D$ as the vertex set, with $l_V(v) = v$. Join the vertices by edges until the graph is connected, whilst maintaining planarity. Embed this graph. Now add additional vertices and edges to the embedded graph, again whilst maintaining planarity, until all vertices next to the infinite face are labelled $\emptyset$; this graph is an embedding of a connected, planar, vertex-labelled graph that generates an Euler diagram with abstraction D.

Theorem 5. *Let D be an abstract description. Then there exists a connected, planar, vertex-labelled graph, G with an appropriate embedding, $\hat{G}$, that generates an Euler diagram with abstraction D.*

Consequently, our method is sufficiently general that we are guaranteed to be able to embed any abstract description. This is not the case for the majority of the previously existing methods in the literature. We are also able to describe exactly which class of Euler diagrams can be generated using our method, captured in Theorems 6 and 7. First, we define *minimal regions*.

Definition 18. A **minimal region** of an Euler diagram, $d = (Curve, l)$, is a connected component of $\mathbb{R}^2 - \bigcup_{c \in Curve} im(c)$.

It follows that a zone is simply a union of minimal regions.

Definition 19. Let d be an Euler diagram such that

(1) d has a finite number of minimal regions,
(2) no curve in d runs self-concurrently, and
(3) for any contour in d, no pair of its curves run concurrently.

Then we say that d is **generatable**.

Theorem 6. *Let d be a generatable Euler diagram. Then there exists a connected, planar, vertex-labelled graph that generates d.*

Theorem 7. *Let G be a connected, planar, vertex-labelled graph. Then any Euler diagram generated from G is generatable.*

From a visualization perspective, this means that our method generates the vast majority of diagrams one would want to consider (one would not be able to accurately visualize an Euler diagram with an infinite number of minimal regions, for instance).

From Theorems 6 and 7, it follows that if an abstract description can be realized as a generatable diagram that possesses some specified collection of the six properties described in Section 5 then our method is able to generate such a diagram.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

5. Properties of graphs and Euler diagrams

In this section, we will show how we can use an embedding of a vertex-labelled graph, $\hat{G}$, to identify properties possessed by a generated Euler diagram, d. To recap, the following are properties that Euler diagrams may possess; they are formalized in [27], but for completeness we include their formalization in the relevant subsections below.

Definition 20. Given an Euler diagram $d = (Curve, l)$, the following are properties that d may posses.

(1) **No concurrency** No pair of curves in $Curve$ run concurrently.
(2) **Connected zones** Each zone in d is connected (i.e. consists of exactly one minimal region).
(3) **No *n*-points** There are no n-points of intersection between the curves in $Curve$.
(4) **Simplicity** All of the curves in $Curve$ are simple.
(5) **Unique labels** The labelling function, l, is injective.
(6) **No brushing points** There are no brushing points between any curves in $Curve$.

In each case, we identify properties possessed by the graph $\hat{G}$ that correspond to the properties possessed by d. Using the properties of $\hat{G}$ we can also count the number of times d violates each of the six properties. The ability to count violations has advantages from a generation perspective. It is computationally hard to find a graph that generates a diagram possessing a specified collection of properties. Current graph based methods for generating diagrams from any abstract description take exponential time in the worst case [19]. Similarly our generation method has exponential time worst case performance relative to the number of zones in the abstract description. The computationally expensive part of the search is through the space of vertex-labelled graphs, if we want to embed an Euler diagram with particular properties. If we do not mind which properties our embedded Euler diagram possess, then we have already provided a trivial method of producing an appropriate vertex-labelled graph. From an implementation perspective, after finding a vertex-labelled graph, we only need to then find a dual of this graph which can be done in polynomial time.

It may be that certain properties are desirable, rather than necessary, for instance. Therefore, we might be prepared to find a graph that generates a diagram with a small number of violations for a particular property rather than no violations at all. Indeed, not all abstraction descriptions can be embedded under particular sets of properties. This means that sometimes we will necessarily have to choose graphs that generate diagrams with particular violations. We can utilize the ability to count violations in heuristic searches to choose vertex-labelled graphs from which to generate diagrams. We further discuss below the use of heuristics to search through the space of vertex-labelled graphs in order to reduce the amount of computation involved in finding a suitable vertex-labelled graph.

5.1. No concurrency

A diagram possesses no concurrency when none of its curves run concurrently. In the following definition (and at other places in the paper) we assume that the domain of each curve in our Euler diagram is $[0, 1]$ in order to simplify the notation.

Definition 21. An Euler diagram, $d = (Curve, l)$, possesses the **no concurrency property** if for all $c_1, c_2 \in Curve$,

(1) if $c_1 \neq c_2$ then $image(c_1) \cap image(c_1)$ is a discrete set of points (equivalently, $\{x \in \mathbb{R}^2 : \exists a, b \in [0, 1]\, c_1(a) = c_2(b) = x\}$ is a discrete set of points) and
(2) if $c_1 = c_2$ then

$$\{x \in \mathbb{R}^2 : \exists a, b \in [0, 1]\, a \neq b \wedge c_1(a) = c_1(b) = x\}$$

is a discrete set of points.

Our generation method makes it very easy to identify when a graph gives rise to an Euler diagram that possesses the no concurrency property, as follows. Consider an edge, e^*, in a dual, $\hat{G}^*$, of an embedded vertex-labelled graph, G. Now, e^* forms part of curves in d whose label is in $l_E(e^*)$. Thus, d has concurrent curves precisely when $\hat{G}^*$ contain an edge, e^*, with $|l_E(e^*)| > 1$. But the edge e^* is labelled the same as the edge, e, with which e^* intersects. Therefore, d contains concurrent curves whenever $\hat{G}$ contains an edge with more than one curve label in its label. Since the edge labelling of $\hat{G}$ is independent of its embedding, we can also detect the presence of concurrency from G.

Lemma 4. *Let $G = (V, E, l_V, l_E)$ be a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. Then two distinct curves, c_1 and c_2, in $Curve$ run concurrently if and only if there exists an edge, $e \in E$, such that the label of e includes the labels of c_1 and c_2, that is $l(c_1) \in l_E(e)$ and $l(c_2) \in l_E(e)$.*

Theorem 8. *Let $G = (V, E, l_V, l_E)$ be a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$. Any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the no concurrency property if and only if for all edges, $e \in E$, the label of e contains at most one label, $|l_E(e)| \leq 1$.*

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

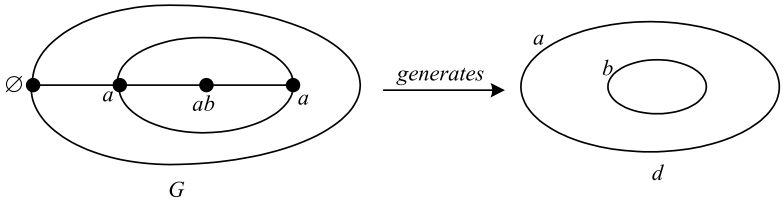

Fig. 17. A diagram with connected zones but non-unique vertex labels in $\hat{G}$.

Definition 22. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. The **concurrency measure** for $\hat{G}$ is

$$\sum_{e \in E} \max(|l_E(e)| - 1, 0).$$

The concurrency measure counts the number of concurrent curve segments in any Euler diagram generated from an appropriate embedding of $\hat{G}$. Thus, if we want to avoid or minimize concurrency, we can aim to find a graph from which to embed with a low concurrency measure.

5.2. Connected zones

A zone is connected whenever it consists of exactly one minimal region.

Definition 23. An Euler diagram, d, possesses the **connected zones property** if all of the zones of d are also minimal regions of d, that is $Z(d) = M(d)$.

When considering connected zones, we first observe that each vertex, v, of $\hat{G}$ is embedded in some face of $\hat{G}^*$. Moreover, v is embedded in a zone, z, whose abstraction is $l_V(v)$, as shown in Theorem 1. This might lead one to believe that d has connected zones precisely when vertices of $\hat{G}$ (equivalently, G) have unique labels. However, this is not the case. If $\hat{G}$ has unique vertex labels then d will have connected zones. There are examples of graphs which do not have unique vertex labels that generate diagrams with connected zones.

To illustrate, the graph in Fig. 17 has two vertices labelled a but the diagram it generates possesses the connected zone property. We observe that if two vertices, v_1 and v_2, have the same label and are connected by an edge, e, then e is labelled $\emptyset$. In turn, this implies that an edge in $\hat{G}^*$ that intersects e does not form part of any curve in d. From this, it follows that v_1 and v_2 are in the same minimal region of d.

Lemma 5. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$ and let z be a zone in d. Then z is connected if and only if the graph $\hat{G}_V(ab(z))$ is connected.*

Proof. Let z be a zone in d and suppose that z is connected. Then z consists of exactly one minimal region, m. Therefore each vertex, v, in $\hat{G}$ with label $l_V(v) = ab(z)$ is embedded in $m = z$, by Corollary 1. Suppose that there are two vertices, v_1 and v_2, embedded in z but for which there is no path in $\hat{G}_V(ab(z))$ from v_1 to v_2, as illustrated in Fig. 18. Now, in $\hat{G}^*$, the dual of $\hat{G}$ that gave rise to d, v_1 and v_2 are embedded in different faces of $\hat{G}^*$ (by the definition of the dual). Therefore, there must be a set of edges in $\hat{G}^*$ embedded entirely within z that split z into two components, com_1 and com_2 which contain v_1 and v_2 respectively,[2] as shown in Fig. 19. Since z is connected, these edges are all labelled $\emptyset$. Now, this means that there must be a path, p, from v_1 to some vertex, v_3, of $\hat{H}$ embedded in com_2, as shown in Fig. 20. But then one can show that com_2 is further subdivided in to two components (following the same strategy), with v_3 joined to a vertex, v_4 in the same component as v_2 and so forth, see Fig. 21. Since there are only a finite number of vertices (of $\hat{G}$) embedded in z, eventually this process of subdividing components will show that there is a path from v_1 to v_2, contradicting our initial assumption that no such path existed. Hence, if z is connected, the graph $\hat{G}_V(ab(z))$ is connected.

For the converse, suppose that $\hat{G}_V(ab(z))$ is connected. Let v_1 and v_2 be vertices in $\hat{G}_V(ab(z))$ that are joined by an edge, e. Then the edge, e^*, of $\hat{G}^*$, that crosses e is labelled $\emptyset$. This means that no curve in d traverses e^* and, therefore, v_1 and v_2 are in the same minimal region of d. Since $\hat{G}_V(ab(z))$ is connected, it follows that all vertices in $\hat{G}_V(ab(z))$ lie in the same minimal region of d. By Corollary 1, and the definition of $\hat{G}_V(ab(z))$, this minimal region is the zone with abstraction $ab(z)$. Hence z is connected. □

[2] Note: these edges need not form a path since z may not be simply connected.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

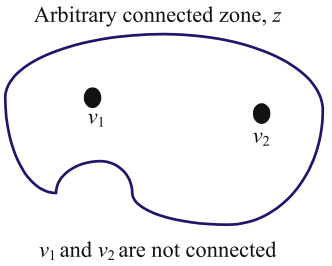

Fig. 18. A connected zone in which v_1 and v_2 are not connected.

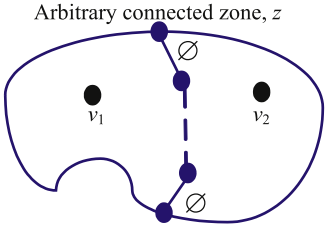

Fig. 19. A path of edges across z separating v_1 and v_2.

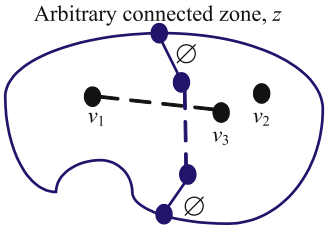

Fig. 20. The vertex v_1 must be joined to v_3.

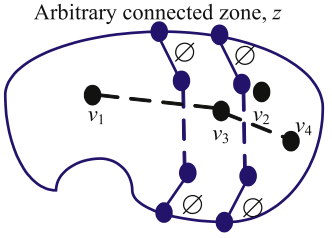

Fig. 21. The vertex v_3 must be joined to v_4.

Theorem 9. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the connected zones property if and only if for all $az \in im(l_V)$, the graph $\hat{G}_V(az)$ is connected.*

As with concurrency, we can detect the connectedness of zones from G, since the arguments above are independent of the actual embedding $\hat{G}$. To count the number of minimal regions of which a zone, z, consists, we can count the number of components in $G_V(ab(z))$, the subgraph of G that contains all and only vertices with label $ab(z)$. The following disconnected zones measure, therefore, counts the number of 'extra' minimal region of which each zones consist (each zone consists of at least one minimal region).

Definition 24. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. The **disconnected zones measure** for $\hat{G}$ is

$$\sum_{az \in im(l_V)} (|comp(\hat{G}_V(az))| - 1).$$

5.3. No n-points

An n-point in a diagram is one that is mapped to n times by the curves.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

Definition 25. Let $d = (Curve, l)$ be an Euler diagram and let p be a point in $\mathbb{R}^2$. We say that p is an ***n*-point** in d if $\sum_{c \in Curve} \left|\{x \in [0, 1) : c(x) = p\}\right| \geq n$.

Definition 26. An Euler diagram possesses the **no *n*-points property** provided each point in $\mathbb{R}^2$ is at most an $(n-1)$-point in d.

Unlike concurrency and disconnected zones, the presence of n-points needs to be determined from an actual embedding, $\hat{G}$. Given an embedding, however, it is relatively straightforward to identify n-points. In a generated Euler diagram, an n-point arises from a vertex, v, in the dual graph, $\hat{G}^*$, where v is incident with edges that, between them, contain at least $2n$ occurrences of labels. This can be translated into a property of $\hat{G}$ by appealing to edge sequences around faces. Before we proceed, it is useful to denote the bag of edges incident with v that includes each non-loop once and each loop twice by $incident(v)$.

Theorem 10. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the no n-points property if and only if for each face, f, of $\hat{G}$ with $ES(f) = (e_1, \ldots, e_k)$,*

$$\sum_{1 \leq i \leq k} \frac{|l_E(e_i)|}{2} < n.$$

Proof. Suppose that d possesses the no n-points property. Let f be a face of $\hat{G}$. The vertex, v, of $\hat{G}^*$ embedded in f forms an m-point in d, where

$$m = \sum_{e \in incident(v)} \frac{|l_{E^*}(e)|}{2}.$$

Since there is a bijective correspondence between $incident(v)$ and the edges in $ES(f)$ that preserves edge labelling, it follows that

$$m = \sum_{1 \leq i \leq k} \frac{|l_E(e_i)|}{2}.$$

Therefore, since $m < n$, each face, f, of $\hat{G}$ satisfies

$$\sum_{1 \leq i \leq k} \frac{|l_E(e_i)|}{2} < n.$$

Conversely, suppose that each face, f, of $\hat{G}$ satisfies

$$\sum_{1 \leq i \leq k} \frac{|l_E(e_i)|}{2} < n$$

for some n. Suppose that d possesses the m-point property but not the $(m-1)$-point property. We must show that $m < n$. Clearly, a point, p, in $\bigcup_{c \in Curve} im(c)$ that is an m-point lies on a vertex, v, of $\hat{G}^*$. The method used to convert the edges of $\hat{G}^*$ into curves ensures any point such as p is an $\sum_{e \in incident(v)} \frac{|l^*_E(e_i)|}{2}$-point. But

$$\sum_{e \in incident(v)} \frac{|l^*_E(e_i)|}{2} = \sum_{1 \leq i \leq k} \frac{|l_E(e_i)|}{2} < n$$

as required. □

We can use the above insights to define a measure to count the number of times the n-points measure is violated in a generated Euler diagram.

Definition 27. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. The ***n*-points measure** for $\hat{G}$ is

$$\sum_{f_x \in face(\hat{G})} \frac{\max\left(\left(\sum_{1 \leq i \leq k_x} |l_E(e_i)|\right) - 2(n-1), 0\right)}{2}$$

where $face(\hat{G})$ is the set of faces in $\hat{G}$ and $ES(f_x) = (e_1, \ldots, e_{k_x})$ is some edge sequence around f.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

5.4. Simple curves

The detection of non-simple curves also requires an embedding of G. Non-simplicity arises in d when a curve passes through a vertex of $\hat{G}^*$ more than once. As shown above, we can always decompose a non-simple curve into a set of simple curves. We detect whether, if no decomposition takes place, the curves in the generated diagram are simple. We start by noting that a contour that consists of entirely simple curves, for which no pair intersect, is called a *simple contour*. Under our generation method (without any decomposition), all of the curves are simple if and only if all of the contours are simple. Firstly, we formalize the simplicity property.

Definition 28. An Euler diagram, $d = (Curve, l)$, possesses the **simplicity property** if and only if all of the curves in $Curve$ are simple.

Lemma 6. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$ and let $Con(L)$ be a contour in d. Then $Con(L)$ is simple if and only if for each face f_x of $\hat{G}$, where $ES(f) = (e_1, \ldots, e_{k_x})$ is some edge sequence around f_x, we have*

$$\sum_{1 \leq i \leq k_x} (l_E(e) \cap \{L\}) \leq 2.$$

Proof. Let $Con(L)$ be a simple contour in d. Then every face of $\hat{G}$ contains a vertex, v, of $\hat{G}^*$ that is incident with at most two edges that contain the label L (otherwise $Con(L)$ would not be simple). Since there is a bijection between $incident(v)$ and those in $ES(f)$ which preserves labelling, it follows that

$$\sum_{1 \leq i \leq k} (l_E(e) \cap \{L\}) \leq 2.$$

For the converse, suppose that $\sum_{1 \leq i \leq k} (l_E(e) \cap \{L\}) \leq 2$. Then the vertex, v, in $\hat{G}^*$ embedded in f is incident with zero, one, or two edges that contain L. In the first case, $Con(L)$ does not pass through v. In the second case, the curve, c, labelled L passing through v is essentially a loop and, therefore, simple. In the third case c passes through v exactly once (by the construction of c from the edges of $\hat{G}^*$. Clearly, the only places that a contour can self-intersect are at the vertices of $\hat{G}^*$. Therefore $Con(L)$ is simple). □

The following theorem then follows trivially.

Theorem 11. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the simplicity property if and only if for each label, $L \in \bigcup_{z \in im(l_V)} z$, and for each face, f, in $\hat{G}$, with $ES(f) = (e_1, \ldots, e_k)$,*

$$\sum_{1 \leq i \leq k} (l_E(e) \cap \{L\}) \leq 2.$$

The non-simplicity measure counts the number of times each curve self-intersects in a generated diagram, d. If we decompose the curves of d then the measure below counts the number of times each contour self-intersects. That is, the measure is invariant under curve decomposition.

Definition 29. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. The **non-simplicity measure** for $\hat{G}$ is

$$\sum_{L \in im(l_V)} \left(\sum_{f_x \in face(\hat{G})} \max \left(\left(\sum_{1 \leq i \leq k_x} \frac{|l_E(e_i) \cap \{L\}| - 2}{2}, 0 \right) \right) \right)$$

where $face(\hat{G})$ is the set of faces in $\hat{G}$ and $ES(f_x) = (e_1, \ldots, e_{k_x})$ is some edge sequence around f_x.

5.5. Unique labels

Again, when detecting whether our embedded diagram possesses unique labels, we assume that no curve decomposition has taken place. However, to avoid any confusion, we will call the measure we derive at the end of the section the contour components measure. This is because our measure counts the number of components of which a contour consists, regardless of whether any decomposition has been performed.

Definition 30. An Euler diagram, $(Curve, l)$, possesses the **unique labelling property** if the function l is injective.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

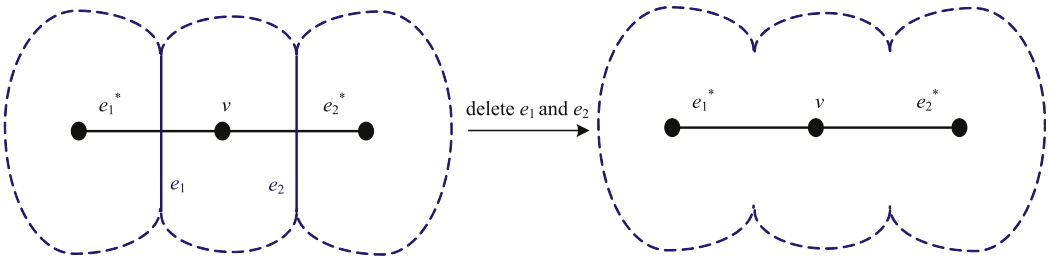

Fig. 22. Connected edges are in a common face.

Lemma 7. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let $d = (Curve, l)$ be an Euler diagram generated from $\hat{G}$. Let $L \in im(l)$. Then $Con(L)$ consists of a single curve if and only if there exists a face, f, of the graph $\hat{G} - \{e \in E : L \in l_E(e)\}$ such that all of the edges in $\hat{G}_E(L)$ are embedded in f.*

Proof. Suppose that $Con(L)$ consists of a single curve. This implies that $\hat{G}_E^*(L)$ is connected. Consider the graph $\hat{G} - \{e \in E : L \in l_E(e)\}$. We note that the edges of $\hat{G}_E(L)$ are embedded in f if and only if all of the edges of $\hat{G}_E^*(L)$ are embedded in f. Let e_1^* and e_2^* be edges of $\hat{G}_E^*(L)$. We show that e_1^* and e_2^* are embedded in the same face of $\hat{G} - \{e \in E : L \in l_E(e)\}$. Suppose first that e_1^* and e_2^* are both incident with a common vertex, v^*. Then v^* is in a face of $\hat{G}$ bounded by edges which include e_1^* and e_2^*, say e_1 and e_2 respectively. Since e_1^* and e_2^* both contain L in their labels, so do e_1 and e_2. This implies that e_1 and e_2 are not in the graph $\hat{G} - \{e \in E : L \in l_E(e)\}$. Moreover, deleting e_1 and e_2 from $\hat{G}$ places the edges e_1^* and e_2^* in the same face of the resulting graph; see Fig. 22, where the dashed lines indicate the edges of $\hat{G}$ that bound the faces containing the vertices incident with e_1^* and e_2^*. Therefore, e_1^* and e_2^* are in some common face, f, of $\hat{G} - \{e \in E : L \in l_E(e)\}$. Since $\hat{G}_E^*(L)$ is connected, it follows that all of its edges are embedded in f and so, therefore, are those of $\hat{G}_E(L)$.

Conversely, suppose that there exists a face, f, of the graph $\hat{G} - \{e \in E : L \in l_E(e)\}$ such that all of the edges in $\hat{G}_E(L)$ are embedded in f. To prove that $Con(L)$ consists of a single curve, it is sufficient to show that the graph $\hat{G}_E^*(L)$ is connected. Suppose that v_1^* and v_2^* are edges of $\hat{G}_E^*(L)$ but for which there is no path in $\hat{G}_E^*(L)$ from v_1^* to v_2^*. The argument to show that this leads to a contradiction is similar to that constructed in Lemma 5. Hence d possesses the unique labels property. □

Theorem 12. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the unique labels property if and only if for all $L \in im(l)$ there exists a face, f, of the graph $\hat{G} - \{e \in E : L \in l_E(e)\}$ such that all of the edges in $\hat{G}_E(L)$ are embedded in f.*

Definition 31. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. The **contour components measure** for $\hat{G}$ is

$$\sum_{L \in im(l_V)} (|face(\hat{G} - \{e \in E : L \in l_E(e)\}, L)| - 1)$$

where $face(\hat{G} - \{e \in E : L \in l_E(e)\}, L)$ is the set of faces in $\hat{G} - \{e \in E : L \in l_E(e)\}$ that contain at least one edge, e, of $\hat{G}$ where $L \in l_E(e)$.

5.6. No brushing points

Detecting the presence of brushing points is the most challenging case, out of all six properties. In fact, we only present partial results here, in that we can sometimes detect such points. In Fig. 23, the generated diagram d has a brushing point between a and c. This brushing point arises from the vertex of $\hat{G}^*$ that is embedded in f (the figure does not show $\hat{G}^*$). Each edge sequence around f gives rise to a label sequence (reading the labels of the edges rather than the edges themselves): (c, a, a, c). This tells us that curves labelled a and c embedded across f will not cross. If that edge sequence had instead given rise to (c, a, c, a) then this would indicate that the curves labelled a and c cross in f. In general, edges are labelled by sets of curve labels, rather than single curve labels, and these two sequences are strictly $(\{c\}, \{a\}, \{a\}, \{c\})$ and $(\{c\}, \{a\}, \{c\}, \{a\})$.

To formalize the notion of a brushing point, we also consider when two curves cross. As an example, the curves in the left-hand diagram in Fig. 24 cross at the point p. To identify this formally, we can consider a disc neighbourhood $N(p)$ around p: it contains four regions, one for each of the four combinations of being 'inside' or 'outside' c_1 and c_2. However, had c_1, say, possessed self-concurrency it need not have had an interior. Thus, we cannot use the notion of interior to define a crossing point in general Euler diagrams. Given c_1, $N(p)$ is cut into two pieces and we can arbitrarily assign positive and negative to each of these two pieces respectively. The curves in the right-hand diagram brush at p. Here, the positive and negative regions do not correspond to each of the four combinations of being positive or negative to the two curves.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

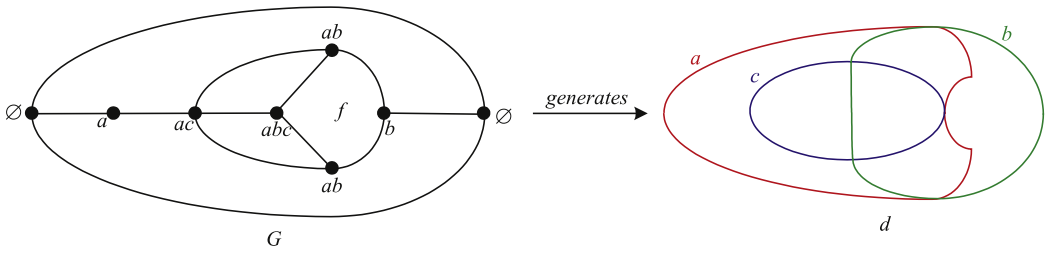

Fig. 23. Detecting transverse crossings.

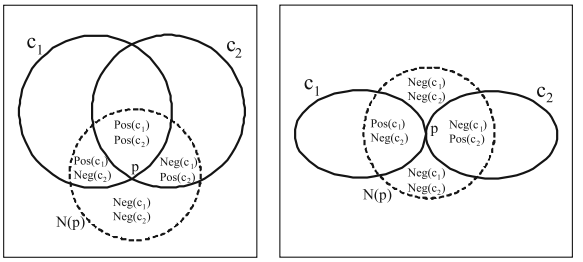

Fig. 24. Crossing and brushing curves.

Definition 32. Let $c\colon [x, y] \to \mathbb{R}^2$, where the notation $[x, y]$ denotes any closed interval of $\mathbb{R}^2$. Let p be a point in $im(c)$ and let $N(p)$ be a disc neighbourhood of p. If $N(p) - im(c)$ consists of exactly two connected components then $N(p)$ is called a **splitting neighbourhood** of p for c.

Definition 33. Let $c_1\colon [x, y] \to \mathbb{R}^2$ and $c_2\colon [a, b] \to \mathbb{R}^2$ be two curves such that there is a unique point p in $im(c_1) \cap im(c_2)$. If there exists a disc neighbourhood $N(p)$ such that

(1) $N(p)$ is a splitting neighbourhood of p for c_1; arbitrarily call one of the two connected components $Pos(c_1)$ and the other $Neg(c_1)$,
(2) $N(p)$ is a splitting neighbourhood of p for c_2; arbitrarily call one of the two connected components $Pos(c_2)$ and the other $Neg(c_2)$,
(3) $N(p)$ contains a point in $Pos(c_1) \cap Pos(c_2)$,
(4) $N(p)$ contains a point in $Pos(c_1) \cap Neg(c_2)$,
(5) $N(p)$ contains a point in $Neg(c_1) \cap Pos(c_2)$ and
(6) $N(p)$ contains a point in $Neg(c_1) \cap Neg(c_2)$

then c_1 and c_2 are said to **cross** at p. Otherwise c_1 and c_2 **brush** at p.

Let $c_1\colon [0, 1] \to \mathbb{R}^2$ and $c_2\colon [0, 1] \to \mathbb{R}^2$ be two closed curves such that there is a point p in $im(c_1) \cap im(c_2)$. If there exists $I_1, I_2 \subseteq [0, 1]$ such that

(1) $im(c_1|_{I_1}) \cap im(c_2|_{I_2})$ contains exactly p,
(2) the curves $c_1|_{I_1}$ and $c_2|_{I_2}$ brush at p

then c_1 and c_2 **brush** at p.

Definition 34. An Euler diagram, d, possesses the **no brushing points property** if and only if no pair of curves brush at any point.

Definition 35. Let G be a connected, planar, vertex-labelled graph with an appropriate embedding $\hat{G}$. Let f be a face in $\hat{G}$ with an edge sequence $ES(f) = (e_1, \ldots, e_n)$. Then the **label sequence** given labels L_1 and L_2 in $\mathcal{L}$, derived from $ES(f)$ is $(l_E(e_1) \cap \{L_1, L_2\}, \ldots, l_E(e_n) \cap \{L_1, L_2\})$. The **reduced label sequence**, $(a_1, \ldots, a_n)$, derived from $ES(f)$ is the maximal sequence obtained from $(l_E(e_1) \cap \{L_1, L_2\}, \ldots, l_E(e_n) \cap \{L_1, L_2\})$ by removing all occurrences of $\emptyset$. Given a reduced label sequence, a **brushing test sequence** for f is $(b_1, \ldots, b_m)$ where

(1) a_1 gives rise to $b_1 \in a_1$ and if $|a_1| = 2$ then $b_2 = a_1 - \{b_1\}$, and
(2) each remaining a_j gives rise to
    (a) b_x where $b_x \in a_j$, and
    (b) if $|a_j| = 2$, then $b_{x+1} = a_j - \{b_x\}$
    where b_{x-1} arose from a_{j-1}.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

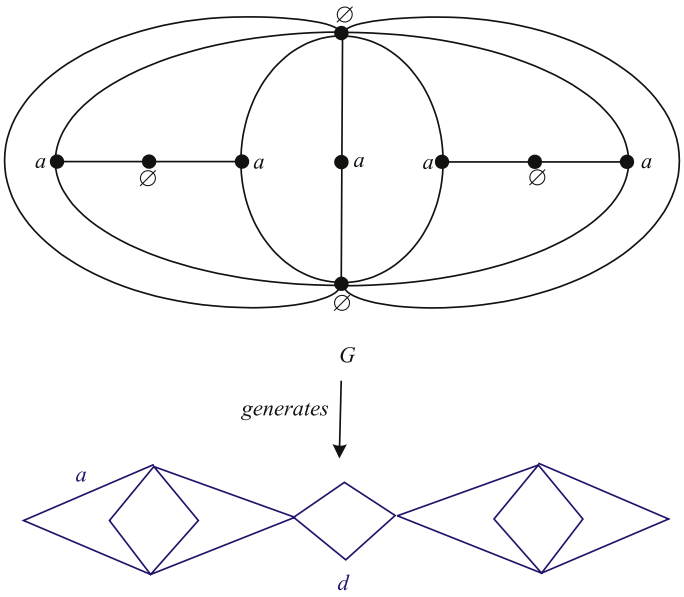

Fig. 25. A graph that does not have detectable brushing points but generates a diagram with a brushing point.

To illustrate, a face edge sequence $(\{a, c\}, \{a, b\}, \{b\}, \{c\})$ has a label sequence, given a and b, $(\{a\}, \{a, b\}, \{b\}, \{\})$. In turn, this gives rise to the reduced label sequence $(\{a\}, \{a, b\}, \{b\})$. From this, we can generate two brushing test sequences: (a, a, b, b) and (a, b, a, b); the set $\{a, b\}$ gives rise to the middle two labels in the sequence, one a and one b, in either order. We can use brushing test sequences to (partially) test for the presence of brushing points.

Definition 36. Let $G = (V, E, l_V, l_E)$ be a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let f be a face in $\hat{G}$. Let L_1 and L_2 be labels in $\bigcup_{az \in im(l_V)} az$. If, for all brushing test sequences, $b = (b_1, \ldots, b_m)$, for f given L_1 and L_2, there exists a subsequence $b' = (b_y, b_{y+1}, \ldots, b_{y+\frac{m}{2}-1})$ where the number of occurrence of L_1 in b' is greater than half the number of occurrences of L_1 in b then f is said to have a **detectable brushing point** for L_1 and L_2.

Theorem 13. *Let $G = (V, E, l_V, l_E)$ be a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$. Then in any Euler diagram that is generated from $\hat{G}$, two curves, c_1 and c_2, in d meet at a brushing point in a face f of $\hat{G}$ if f has a detectable brushing point for $l(c_1)$ and $l(c_2)$.*

Unfortunately, a graph may have faces that have no detectable brushing points for labels L_1 and L_2 but curves passing through that face with those labels brush. However, this only happens when we have a curve passing through a face multiple times (i.e. that curve has a non-simple point in that face). To illustrate, in Fig. 25, the graph $\hat{G}$ has faces that have no detectable brushing points, given a (i.e. both L_1 and L_2 are a). However, no matter how we traverse the curve a, shown in the generated diagram below, we will create a brushing point. In a more complex example, we might have two curve labels a and b for which some face has a brushing test sequence (a, b, a, a, b, a). From such a sequence we cannot detect whether a and b cross or brush.

Theorem 14. *Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. If for any pair of labels, L_1 and L_2, in $im(l_V)$ and for any face f_x in $\hat{G}$,*

(1) *f_x has no detectable brushing point for L_1 and L_2,*
(2) *$\sum_{1 \le i \le k_x} (l_E(e) \cap \{L_1\}) \le 2$ where $ES(f_x) = (e_1, \ldots, e_{k_x})$ (a condition which is equivalent to saying the curves with label L_1 are simple), and*
(3) *similarly for L_2, $\sum_{1 \le i \le k_x} (l_E(e) \cap \{L_2\}) \le 2$*

then any Euler diagram, $d = (Curve, l)$, generated from $\hat{G}$ possesses the no brushing points property.

Our brushing points measure will provide a lower bound on a count of the number of times an embedded diagram d violates this property. It is possible for the measure to be zero and yet d may not possess the no brushing points property.

Definition 37. Let $G = (V, E, l_V, l_E)$ be a connected, planar, vertex-labelled graph with an appropriate embedding, $\hat{G}$. Let L_1 and L_2 be labels in $\bigcup_{az \in im(l_V)} az$. The **brushing points score** for L_1 and L_2 is the number of faces in $\hat{G}$ for which there is a detectable brushing point for L_1 and L_2.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

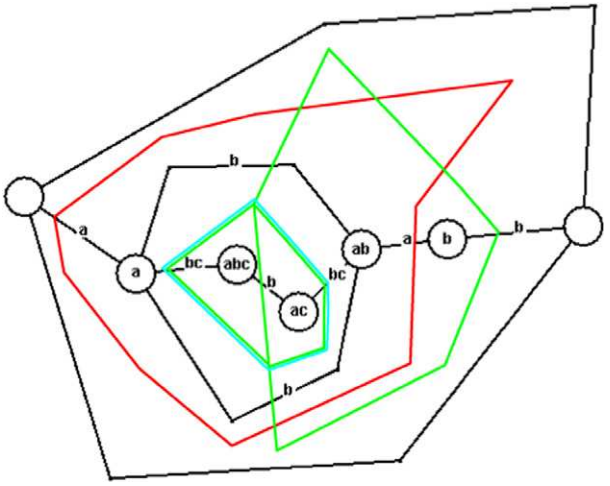

Fig. 26. An incorrect embedding for a dual of a vertex-labelled graph.

Definition 38. Let $G = (V, E, l_V, l_E)$ be a connected, planar vertex-labelled graph with an appropriate embedding, $\hat{G}$. The **brushing points measure** for G is the sum of the brushing point scores[3] for all pairs of labels L_1 and L_2.

6. Producing vertex-labelled graphs

So far, we have mainly considered how to covert graphs into Euler diagrams and how to use those graphs to identify properties that will be possessed by those diagrams. Largely, we have not discussed how to convert abstract descriptions into vertex-labelled graphs. This is a key step in the generation process and, when proving that every abstract description, D, can be realized as an Euler diagram, d, we illustrated a method to convert D into a vertex-labelled graph that generates such a d.

Here, we sketch a naive algorithm to produce a wider class of vertex-labelled graphs from which we can embedded Euler diagrams. First, take an arbitrary set of vertices, V, with the same cardinality as D. Define an arbitrary bijection from V to D, labelling each vertex with an abstract zone. Clearly the graph, G, with this vertex set and labelling function is a subgraph of any vertex-labelled graph that generates an Euler diagram with abstraction D. We can add edges and vertices to this graph in order to generate all vertex-labelled graphs with up to some arbitrary finite number of vertices. Given a connected, planar graph in this vertex set, we can embedded it in all possible ways and discard any embeddings that are not appropriate. For each of these embeddings, we can determine what properties the embedded Euler diagram will possess.

Thus, if we want to generate an Euler diagram, d, with abstraction D and some specified set of properties, we can search through this finite set of graphs to see whether any generate such a d. Of course, this naive search algorithm is likely to be infeasible to implement in practice, due to the very large search space we are likely to generate. This issue is further discussed in the conclusion, in the context of heuristic searches through the space, an area in which we have already produced some results [24].

7. Implementation

We now describe prototype software that implements a method for embedding an Euler diagram from a vertex-labelled graph. This implementation relates to Section 4, which described how a diagram can be formed from a subgraph of a dual of the vertex-labelled graph. The formation of the abstract dual is not problematic: there is a dual vertex for each face in the vertex-labelled graph and a dual edge for each edge in the vertex-labelled graph. Hence the focus of this section is on routing the edges in the dual in to produce the required diagram.

The routing of the dual edges is not trivial. Firstly, the dual edges must remain in the face between cutting (the vertex-labelled graph) $\hat{G}$'s edges and joining to the vertices of $\hat{G}^*$, otherwise inadvertent edge crossings could occur, leading to a non-plane dual and an incorrectly embedded diagram. Even when routing edges in the same face, inadvertent crossings of dual edges can appear. This is illustrated in Fig. 26: the edge crossings in the topmost face mean the dual is non-plane (the dual vertices are shown as solid rectangles).

Our approach to avoid these difficulties is to triangulate the faces in the vertex-labelled graph. Edge routing can be then be performed by connecting edge segments as straight lines between evenly spaced cut points on triangulation edges. The triangles are concave and, hence, the dual edges cannot leave the triangles, and so cannot leave the face. However, this

[3] We do not count the brushing point score for L_2 and L_1 if we have already counted the brushing point score for L_1 and L_2.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

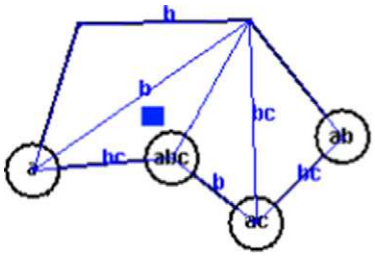

Fig. 27. The triangulated edge (shown unlabelled) needs to be ordered.

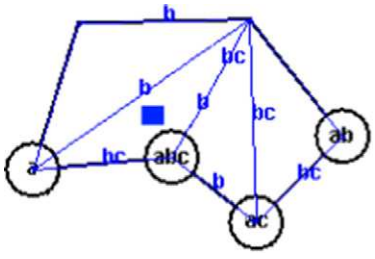

Fig. 28. Ordering labels on dual edges.

means that multiple edges may pass through a triangle, and to ensure that dual edges do not cross inside a face, we can arrange the cuts points for a triangle so that when read around the triangle, no intersections will occur. This is done using an ordering method for the edge cuts along the triangulated edge.

Figs. 27 and 28 show how the dual edge cut points on triangulation edges are found for a single face. Dual edges labelled *b* and *bc* must pass through this triangulated edge. In Fig. 28, these edges have been ordered on the triangulated edge so that they will pass through the triangles without crossing.

In our construction of a dual, we first determine the position of each dual vertex. Given a face of $\hat{G}$, it is placed at the centroid of the triangle which contains the middle of the face (or triangle closest to the centre of the face, in the case of some convex faces). The dual vertex, v, is shown as a solid rectangle in Figs. 27 and 28. The dual edges incident with v must then be routed from the vertex-labelled graph edges to v. There is no need to order cut points through triangulation edges with zero or one dual edges cutting them, so in Fig. 27 the labels for these edges are shown, placed in the middle of the edges. There is one remaining triangulated edge, which has two dual edges cutting it, labelled *b* and *bc*. Such triangulated edges can be ordered, as long as they belong to a triangle that has two edges which have already been assigned an order. It is easy to show that such a triangulated edge will always exist, and so the dual edge routing always terminates. The order that avoids any edge crossings in the triangle not containing the dual vertex can be seen in Fig. 28, as drawing the edge segments as straight lines between the label positions leads to no crossing.

The final Euler diagram, after all dual edge routing, is shown in Fig. 29; the dual vertices are omitted this diagram for clarity. The straight line dual edge segments that join up the cut points on triangulation edges can be seen. The final Euler diagram is given in Fig. 30.

7.1. Choosing a vertex-labelled graph

The implementation described above constructs an Euler diagram from an embedded, vertex-labelled graph. There are challenges in choosing a vertex-labelled graph that generates an Euler diagram with specified properties, or low numbers of violations of properties. It is possible to define heuristic searches through the space of vertex-labelled graphs in order to decrease the time taken to find a suitable graph, at the possible expense of not finding a 'best' graph. We have already developed one such heuristic search and implemented it [24]. That particular search yields a graph that generates Euler diagrams under a prioritized selection of the properties. In particular, that implementation generates diagrams which have connected zones, simple contours (contours for which no pair of curves intersect), and it attempts to minimizes the amount of concurrency and number of brushing points. Here, to indicate the scalability of that implementation, we include some data on the time taken to find a vertex-labelled graph, given an abstract description, that generates such a diagram. The code was written and run using Java 1.6 in 32 bit Vista SP1 on a Intel Core 2 Duo CPU running at 2.4 GHz with 4 GB of RAM. Each randomly generated abstract description had 6 curve labels and between 4 and 23 zones. We randomly generated 10 abstract descriptions with each zone set cardinality (i.e. $4 \leq |Z(D)| \leq 23$), giving a total sample size of 200. As can be seen in Fig. 31, the process is approximately linear in the number of zones; the times given in the chart are the average times taken to find graphs for each of the 10 abstract descriptions.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

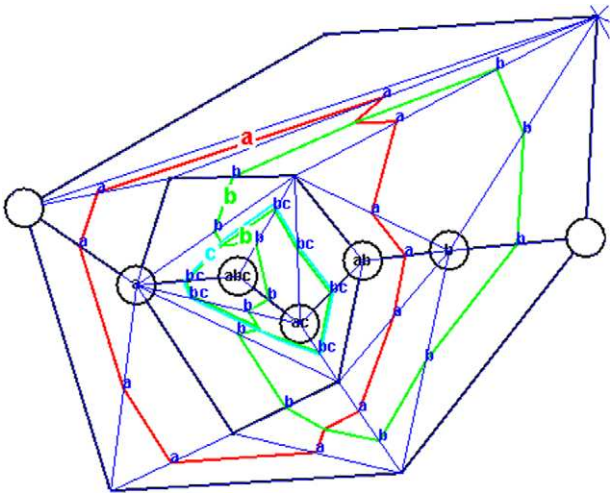

Fig. 29. Vertex-labelled graph, triangulation and Euler diagram drawn from the dual graph.

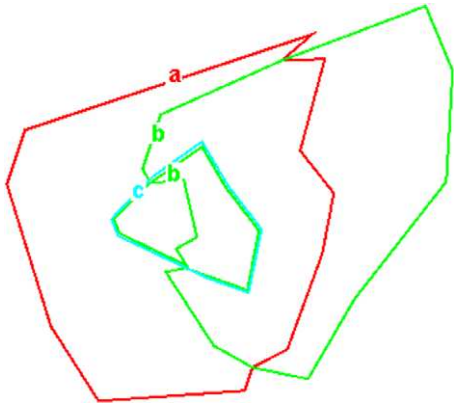

Fig. 30. Final Euler diagram.

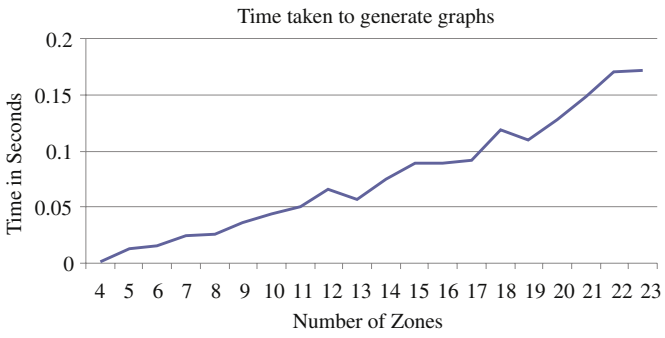

Fig. 31. Generating vertex-labelled graphs.

8. Conclusion and further work

In this paper, we have produced the first fully formalized framework for generating an Euler diagram given any abstract description. Our method takes the abstract description and, first, converts it into a vertex-labelled graph. Secondly, after finding an appropriate embedding we construct a dual graph which allows us to generate an Euler diagram with the required abstraction. We have identified properties of the embedded graph that correspond to properties that the generated Euler diagram will possess. Moreover, we have shown how to use the embedded graph to count violations of these properties in the embedded Euler diagram. A particular feature of our generation method is that it is capable of producing an embedding of any Euler diagram that has curves with only a finite number of self-intersections, contours that are not self-concurrent, and a finite number of minimal regions. Finally, we produced a naive implementation of the method to justify its utility.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005

Further work includes allowing more general edge labelling to occur in vertex-labelled graphs. At present, our graphs do not generate Euler diagrams that include any self-concurrent curves. Whilst there are usability reasons for not allowing such curves (one cannot necessarily identify the interior of such curves from their images alone, for instance) it is a limitation of our method and sometimes drawing such curves can be advantageous. For example, in an automated reasoning environment, such as [28], one may want to draw one diagram so that it looks similar to another (after the application of an inference rule). Such a requirement may necessitate the use of a self-concurrent curve. Each edge label would need to reflect the number of times each curve was to pass through the respective edge.

The work in this paper also provides the essential foundations for further significant avenues of future research. In particular, we have set up a framework that will allow heuristic searches to take place that allow the generation of Euler diagrams that have specified arbitrary subsets of the properties investigated in this paper. As discussed above, we have already begun work along these lines, producing a more sophisticated implementation than that described in this paper [24]. We plan to extend this work to include a user specified prioritization of the properties, which represents a considerable challenge.

Acknowledgements

This work is supported by EPSRC grants EP/E011160/1 and EP/E010393/1 for the Visualization with Euler Diagrams project. The project web site can be found at www.eulerdiagrams.com.

References

[1] L. Euler, Lettres a une princesse dallemagne sur divers sujets de physique et de philosophie, Letters 2 (1775) 102–108, berne, Socit Typographique.
[2] S. Chow, F. Ruskey, Drawing area-proportional Venn and Euler diagrams, in: Proceedings of Graph Drawing 2003, Perugia, Italy, in: LNCS, vol. 2912, Springer-Verlag, 2003, pp. 466–477.
[3] 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) (2005) 1592–1595.
[4] A. Verroust, M.-L. Viaud, Ensuring the drawability of Euler diagrams for up to eight sets, in: Proceedings of 3rd International Conference on the Theory and Application of Diagrams, in: LNAI, vol. 2980, Springer, Cambridge, UK, 2004, pp. 128–141.
[5] R. DeChiara, U. Erra, V. Scarano, VennFS: A Venn diagram file manager, in: Proceedings of Information Visualisation, IEEE Computer Society, 2003, pp. 120–126.
[6] J. Lovdahl, Towards a visual editing environment for the languages of the semantic web, Ph.D. Thesis, Linkoping University, 2002.
[7] P. Hayes, T. Eskridge, R. Saavedra, T. Reichherzer, M. Mehrotra, D. Bobrovnikoff, Collaborative knowledge capture in ontologies, in: Proceedings of the 3rd International Conference on Knowledge Capture, 2005, pp. 99–106.
[8] S.-K. Kim, D. Carrington, Visualization of formal specifications, in: 6th Aisa Pacific Software Engineering Conference, IEEE Computer Society Press, Los Alamitos, CA, USA, 1999, pp. 102–109.
[9] P. Artes, B. Chauhan, Longitudinal changes in the visual field and optic disc in glaucoma, Progress in Retinal and Eye Research 24 (3) (2005) 333–354.
[10] O. Lenz, A. Fornoni, Chronic kidney disease care delivered by us family medicine and internal medicine trainees: Results from an online survey, BMC Medicine 4 (30) (2006).
[11] E. Hammer, Logic and Visual Information, CSLI Publications, 1995.
[12] R. Clark, Failure mode modular de-composition using spider diagrams, in: Proceedings of Euler Diagrams 2004, in: ENTCS, vol. 134, 2005, pp. 19–31.
[13] L. Niebrój, Defining health/illness: Societal and/or clinical medicine? Journal Physiology and Pharmacology 57 (4) (2006) 251–262.
[14] R. DeChiara, U. Erra, V. Scarano, A system for virtual directories using Euler diagrams, in: Proceedings of Euler Diagrams 04, in: Electronic Notes in Theoretical Computer Science, vol. 134, 2005, pp. 33–53.
[15] J. Thièvre, M. Viaud, A. Verroust-Blondet, Using Euler diagrams in traditional library environments, in: Euler Diagrams 2004, in: ENTCS, vol. 134, 2005, pp. 189–202.
[16] S. Chow, F. Ruskey, Towards a general solution to drawing area-proportional Euler diagrams, in: Proceedings of Euler Diagrams, in: ENTCS, vol. 134, 2005, pp. 3–18.
[17] H. Kestler, A. Muller, J. Kraus, M. Buchholz, T. Gress, H. Liu, D. Kane, B. Zeeberg, J. Weinstein, Vennmaster: Area-proportional Euler diagrams for functional go analysis of microarrays, BMC Bioinformatics 9 (67) (2008).
[18] P. Simonetto, D. Auber, Visualise undrawable Euler diagrams, in: 12th International Conference on Information Visualization, IEEE, 2008, pp. 594–599.
[19] S. Chow, Generating and drawing area-proportional Euler and Venn diagrams, Ph.D. Thesis, University of Victoria, 2007.
[20] G. Stapleton, J. Howse, P. Rodgers, L. Zhang, Generating Euler diagrams from existing layouts, in: Layout of (Software) Engineering Diagrams, ECEASST, 2008.
[21] J. Venn, On the diagrammatic and mechanical representation of propositions and reasonings, Phil. Mag.
[22] A.W. Edwards, Venn diagrams for many sets, New Scientist 7 (1989) 51–56.
[23] J. Flower, J. Howse, Generating Euler diagrams, in: Proceedings of 2nd International Conference on the Theory and Application of Diagrams, Springer, Georgia, USA, 2002, pp. 61–75.
[24] P. Rodgers, L. Zhang, A. Fish, General Euler diagram generation, in: International Conference on the Theory and Application of Diagrams, Springer, 2008.
[25] O. Lemon, I. Pratt, Spatial logic and the complexity of diagrammatic reasoning, Machine GRAPHICS and VISION 6 (1) (1997) 89–108.
[26] N. Swoboda, G. Allwein, Using DAG transformations to verify Euler/Venn homogeneous and Euler/Venn FOL heterogeneous rules of inference, Journal on Software and System Modeling 3 (2) (2004) 136–149.
[27] G. Stapleton, P. Rodgers, J. Howse, J. Taylor, Properties of Euler diagrams, in: Proceedings of Layout of Software Engineering Diagrams, EASST, 2007, pp. 2–16.
[28] G. Stapleton, J. Masthoff, J. Flower, A. Fish, J. Southern, Automated theorem proving in Euler diagrams systems, Journal of Automated Reasoning 39 (2007) 431–470.

Please cite this article in press as: G. Stapleton, et al., A graph theoretic approach to general Euler diagram drawing, Theoretical Computer Science (2009), doi:10.1016/j.tcs.2009.09.005