



Efficient on-line algorithms for Euler diagram region computation

Gennaro Cordasco^a, Rosario De Chiara^{a,*}, Andrew Fish^{b,1}

^a *ISISLab, Dipartimento di Informatica ed Applicazioni, Università degli Studi di Salerno, Italy*

^b *School of Computing, Mathematical and Information Sciences, University of Brighton, UK*

ARTICLE INFO

Article history:

Received 10 December 2008

Accepted 14 July 2010

Available online 17 July 2010

Communicated by N. Magnenat-Thalmann

Keywords:

Euler diagrams

Region computation

Diagram generation

ABSTRACT

Euler diagrams are an accessible and effective visualisation of data involving simple set-theoretic relationships. Sets are represented by closed curves in the plane and often have wellformedness conditions placed on them in order to enhance comprehensibility. The theoretical underpinning for tool support has usually focussed on the problem of generating an Euler diagram from an abstract model. However, the problem of efficient computation of the abstract model from the concrete diagram has not been addressed before, despite this computation being a necessity for computer interpretations of user drawn diagrams. This may be used, together with automated manipulations of the abstract model, for purposes such as semantic information presentation or diagrammatic theorem proving. Furthermore, in interactive settings, the user may update diagrams “on-line” by adding and removing curves, for example, in which case a system requirement is the update of the abstract model (without the necessity of recomputation of the entire abstract model). We define the notion of marked Euler diagrams, together with a method for associating marked points on the diagram with regions in the plane. Utilising these, we provide on-line algorithms which quickly compute the abstract model of a weakly reducible wellformed Euler diagram (constructible as a sequence of additions or removals of curves, keeping a wellformed diagram at each step), and quickly updates both the set of curves in the plane as well as the abstract model according to the on-line operations. Efficiency is demonstrated by comparison with a common, naive algorithm. Furthermore, the methodology enables a straightforward implementation which has subsequently been realised as an application for the user classification domain.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Euler diagrams are a simple, yet effective, means of diagrammatically representing set-theoretic relationships. Their simplicity enhances their applicability for certain classes of data visualisation tasks. They are thought to be an effective representation since the properties of the spatial and domain relationships match, and this gives rise to free rides which are extra inferences that the reader obtains without additional cognitive overhead, due to the effectiveness of the representation [22,36]. Notations based on them have been used in applications such as file-system representation [7–9], genetic set relations visualisation [29–32] and for displaying database query results [41,42]. Another major application area of Euler diagrams is as the basis of many diagrammatic logical reasoning systems [17,18,23,25,27,37] which can be used for formal software specification and reasoning [28,26], for instance.

* Corresponding author.

E-mail addresses: cordasco@dia.unisa.it (G. Cordasco), dechiara@dia.unisa.it (R. De Chiara), Andrew.Fish@brighton.ac.uk (A. Fish).

¹ Partially funded by UK EPSRC grant EP/E011160: Visualisation with Euler Diagrams.

For visual languages, a distinction is made between the concrete model (which consists of the actual drawings in the plane) and the abstract model (which encapsulates the essential information, as pertaining to the semantics of the diagram, for example). A concrete model for an Euler diagram, which will be called a concrete Euler diagram, consists of a set of distinguished curves in the plane together with the set of zones that the curves define: these are regions that are inside a set of curves and outside the rest of the curves in the diagram, and they represent a set intersection. An abstract model for an Euler diagram consists of a set of labels (referred to as abstract curves) corresponding to the concrete curves, together with a set of abstract zones which are subsets of the label set, corresponding to the concrete zones; the abstract zone set may be referred to as an abstract description of the diagram.

Within computer applications that are used to manipulate Euler-based diagrams, computations are often performed at the abstract level. For example, the specification of the semantics of a diagram is computed from the abstract model, and recognition of properties such as nesting enables the automatic presentation of associated set-theoretic statements to be made in a more human readable form [21]. Also, diagrammatic logic reasoning systems tend to be built at the abstract level, with diagrammatic inference rules being syntactic transformations of the abstract model that correspond to logical inference. A proof within a diagrammatic logic can be viewed a sequence of abstract diagrams in which consecutive diagrams differ by the application of a diagrammatic inference rule; automated theorem provers have been developed to search for shortest proofs, for instance [38]. The concept of viewing Euler-based diagrams as a sequence of curve additions, termed building sequences, was considered in [16,17] and a similar idea, using animations in metric space proofs in [43,44]; this was useful since a lack of natural ordering causes interpretation difficulties when symbols representing quantifiers are involved. Furthermore, if the diagrams are being used for set-based information visualisation purposes, then changes in data over time, or as a result of some transformation, can also correspond to a sequence of abstract diagrams with appropriate transformations of the abstract model reflecting the data changes; note that in the information visualisation context the transformations of diagrams is likely to be more general than in the diagrammatic logical setting.

In contrast, the users of any related computer applications would not expect to be faced with abstract models but would expect to see a concrete model. For example, the presentation of a shortest proof in a diagrammatic logic system should be a sequence of concrete diagrams; to produce such a sequence the system may apply a generation algorithm (see below) to each abstract model giving rise to a sequence of concrete diagrams. However, this does not tie together subsequent diagrams within the proof and we would like to preserve the user's mental map and not generate wildly different diagram layouts using a normal generation technique, for instance. Even more difficulties arise in an interactive setting. For example, in interactive diagrammatic theorem proving environments a user would expect to construct, or be presented with, concrete diagrams and be able to apply graphical rules such as add or remove curve and be given feedback as to their applicability, and if applicable to produce the output as a concrete diagram. In diagrammatic interfaces such as those developed for resource management purposes, the user would expect to be able to construct concrete diagrams and classify resources by mechanisms such as drag and drop.

Within such applications that require the user to update or input data, the ability of the system to quickly compute the abstract zone set forms an essential part for the production of effective software. Furthermore the ability to update this information on-line, viewing a diagram as a sequence of contour additions and removals, and keeping track of the abstract model under curve addition or removal would save the system from the complete recalculation of the abstract model at each interactive stage of diagram construction, for instance.

One major problem that has been tackled is the diagram generation problem: given an abstract description of the diagram, automatically generate a concrete Euler diagram which has exactly the correct set of zones. A solution to the Euler diagram generation problem was presented in [19,20], under a particular set of wellformedness conditions (topological and geometric conditions imposed on the diagrams in order to assist in human understanding of the diagrams). In [3], using a slightly less strict set of wellformedness conditions, it was shown that the generation problem is NP-complete. Imposing such wellformedness conditions does restrict the class of diagrams that can be drawn, and in [42] it was shown that every abstract diagram with less than 9 curves can be realised by a concrete diagram using an extension of Euler diagrams (ones that can contain "holes"). In [34], a method for generating a union of regions with holes as a concrete diagram for any abstract description of a diagram was provided. Binary topological relations between regions with holes, which have applications in spatial GIS queries, were investigated in [12–14], but their generation was not considered. A complete classification invariant for binary topological relations between two homogeneously 2-dimensional disks in the plane is defined, extending an earlier model, called the 4-intersection, for binary topological relations based on the intersections and interiors of two point sets in a topological space. An algorithm is provided that minimises the number of individual topological relations necessary to describe a configuration completely.

However, little work has been performed in the opposite direction: quickly computing the abstract diagram from a concrete diagram. In [39,40], models of Euler diagrams based on directed acyclic graphs (DAGs) were used and manipulated. Working with the entire DAG is computationally taxing since a DAG effectively stores the set of all zonal regions; zonal regions are regions describable as inside a set of curves and outside a disjoint set of curves where the union of these two sets is not necessarily a partition of the curve set of the diagram (if the union is a partition then the region is called a zone). In fact, the rest of the DAG can be reconstructed from the leaves which are the set of zones in the usual Euler diagram model (see Section 2, or [19,27] for more details). In [5] the problem was approached by using Java area operations which are platform dependent.

Some applications use an extension called area proportional Euler diagrams [4,29,31]: the areas of the zones are proportional to the size of the set intersection they represent. A new approach to the generation problem, and even the area proportional generation problem, could be to quickly automatically generate a diagram using standard geometric shapes such as circles or ellipses and then quickly check whether the zone set is correct. If not, either the re-generation of another diagram could be performed, or the result could be taken as an approximate solution if the application area allowed this option (approximate solutions to the generation problem based on evolutionary techniques are used in [29] for example).

In this paper, we take a new approach to zone (or region) identification which is based on marking a set of points on the curves to encode the zone information and being able to compute intersections of curves quickly. One way to view this would be that we consider a new domain specific data structure which consists of the set of curves of a concrete diagram together with a set of marked points which are in bijective correspondence with the set of zones, as well as the abstract zone set. From a concrete diagram we can compute the marked point set and use this to compute the abstract zone set. However, we also enable the on-line computation of the abstract zone set when the concrete diagram is altered by the addition or removal of a curve which saves the computation of the whole abstract zone set; this is especially useful in interactive editing environments. The applications of this work are widespread, since it can be utilised in any software systems that utilise Euler diagrams or their extensions.

In more detail, we investigate the problem of keeping track of both the concrete Euler diagram and the abstract Euler diagram. That is, given a concrete diagram d , comprising a collection of curves $\mathcal{C}(d)$, compute the collection of abstract zones $\mathcal{Z}(d)$ for $\mathcal{C}(d)$ and also efficiently update this collection upon the addition/removal of curves to/from $\mathcal{C}(d)$. After some preliminaries in Section 2 and some basic results required in Section 3, we first present the usual naive solution in Section 4 and then our efficient solution in Section 5. Their complexity is analysed and compared in Section 5.3, and conclusions presented in Section 6.

Related work on arrangements

A closely related study is the problem of computing the arrangements of Jordan curves in the plane [11] where, given a set of curves in the plane, the plane is partitioned into *vertices*, which are the intersection points of curves; *edges*, which are the connected components of these curves minus the vertices; and *faces*, which are the maximal connected, open sets, of the complement of the curves. The main results of [11] are based on the following (rephrased) result: Let \mathcal{C} be a collection of closed curves in the plane such that: i) each intersection point involves exactly two curves crossing transversely; and ii) each pair of curves intersect in at most s points. For each curve $A \notin \mathcal{C}$ such that $\mathcal{C} \cup A$ is wellformed (cf. Section 2), the maximum number of edges in the faces of \mathcal{C} that are crossed by A is $O(\lambda_{s+2}(|\mathcal{C}|))$, where $\lambda_s(n)$ is the so-called (n, s) Davenport–Schinzel sequence [1] which is characterised as being almost linear in n for each fixed s . This result is generalised in [10] for d -dimensional space.

It may appear that, given a collection of curves $\mathcal{C}(d)$ defining a concrete Euler diagram d , the computation of an abstract diagram for d is easily reducible to the problem of the computation of the arrangement for $\mathcal{C}(d)$. However, there are several significant differences, especially if one considers the on-line problem, such as:

- the computation of vertices, edges and faces is not sufficient to determine the abstraction of a given concrete diagram. In particular, for each face we would need to compute the set of curves describing the zone, which is not so easy to do in general as one would need to compare each face with each of the curves;
- the computation of vertices, edges and faces is not necessary in our domain specific setting, and in fact it is not desirable. First of all since we are not interested in computing faces, our results are not influenced by the Davenport–Schinzel sequence. Moreover, faces are extremely complicated to store efficiently. Indeed, in [11] the authors state that the faces could be maintained by using a circular, ordered list of the vertices of their boundary. However this approach requires several elaborate and complicated techniques (i.e. Jordan sorting [24]) which make the implementation of this strategy rather impractical. On the other hand, our domain specific approach is easy to implement. This has been demonstrated by the development of a user interface for resource management that utilised an initial implementation of our approach [6];
- we are proposing a set of procedures which compute the abstraction of wellformed Euler diagrams (cf. Section 2) which check at run/draw time whether the current diagram is wellformed or not. In particular our algorithms are able to recognise disconnected zones (that is, faces that are described by the same set of curves);
- we adopt an “online” approach in which diagrams are viewed as a sequence of curve additions and removals. For example, Fig. 1 shows the on-line execution of 4 operations (addition or removal of a curve) starting from an empty diagram. Note that other operations such as translation or resizing of a curve can easily be simulated by the addition and removal operations, and so the algorithms will be applicable in a wider context.

In [6] we presented the design and the implementation, in Java, of a library, called EulerVC, which realises the concepts and algorithms discussed here, although only an informal description of the algorithm was presented. By using this library we developed an application, presented in [6] to interactively handle Euler diagrams for the purposes of resource management (allowing users to interactively draw and modify Euler diagrams which permit the storage and retrieval of internet bookmarks, for instance). In this work, we provide the formal algorithms, together with their complexity analysis.

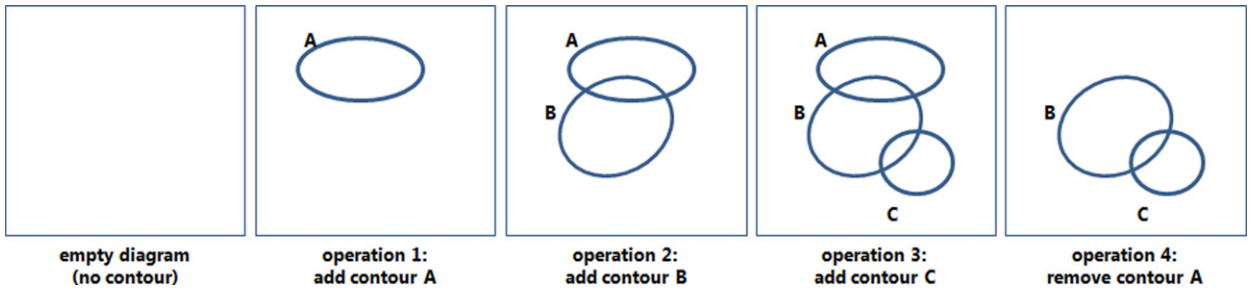


Fig. 1. A sequence of addition and removal operations applied to a concrete diagram.

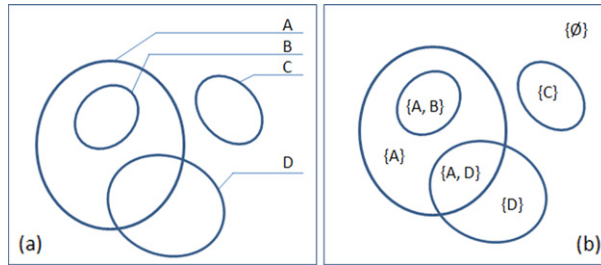


Fig. 2. An example of a wellformed Euler diagram containing 4 curves and 6 zones.

2. Preliminaries

In this section we review some basic definitions and terminology (see [19,27] for more details about the Euler diagram formalisms). Firstly, we introduce notation related to curves in the plane.

Definition 1. Let \mathcal{C} be a collection of simple closed curves in the plane such that:

- (i) each intersection point involves exactly two curves crossing transversely; and
- (ii) each pair of curves intersect in at most s points, for some constant s .

Then the complement of the curves in the plane, denoted $\bar{\mathcal{C}}$, decomposes into non-empty, maximal, connected, open sets, called *minimal regions*. A concrete zone z is a union of minimal regions which is determined by being inside a set of curves $X_z \subseteq \mathcal{C}$ and outside $\mathcal{C} - X_z$. The containing set of curves, X_z , is called the *zone descriptor* for z .

Definition 2. A concrete Euler Diagram (ED) is a pair² $d = \langle \mathcal{C}(d), \mathcal{Z}(d) \rangle$ where:

1. $\mathcal{C}(d)$ is a set of curves,³ in the plane, which intersect in a finite number of points.
2. $\mathcal{Z}(d)$ is the collection of concrete zones z . Formally,

$$z = \left(\bigcap_{c_i \in X_z} \text{int}(c_i) \right) \cap \left(\bigcap_{c_j \in \mathcal{C}(d) - X_z} \text{ext}(c_j) \right),$$

for each $X_z \subseteq \mathcal{C}(d)$, provided this region is non-empty. Here $\text{int}(c)$ and $\text{ext}(c)$ denote the interior and the exterior of c , respectively.

Fig. 2 shows a concrete ED $d = \langle \mathcal{C}(d), \mathcal{Z}(d) \rangle$ with 4 curves named A, B, C, D , and 6 zones described by: $\emptyset, \{A\}, \{A, B\}, \{A, D\}, \{C\}, \{D\}$. Each of the zones is described by its zone descriptor (i.e. the set of its containing curves). For example, the zone descriptor $\{A, D\}$ describes the zone $\text{int}(A) \cap \text{int}(D) \cap \text{ext}(B) \cap \text{ext}(C)$. Notice that the zone which is exterior to all the curves, called the *outer zone* or the *Universe*, is described by \emptyset .

Throughout this paper, we will write \bullet instead of $\bullet(d)$ when the value of d is clear from the context.

A formal definition of an abstract diagram is given below, and so the problem of computing the abstract diagram from a concrete diagram can now be understood. We note that given a concrete diagram, this amounts to simply the computation of the set of all zones descriptors of the diagram and this is what we will compute and track in the algorithms we develop.

² The zone set is included in the definition by tradition, although it is derivable from the curve set.

³ In some works these are required to be labelled but this is unnecessary for our purposes.

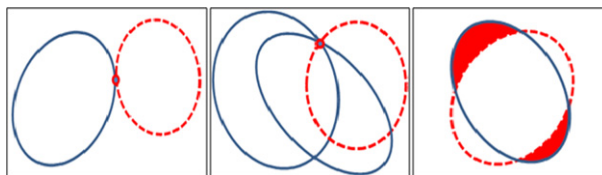


Fig. 3. Examples of not wellformed Euler diagrams.

Definition 3. An *abstract Euler Diagram* (ED) is a pair: $d = \langle \mathcal{L}(d), \mathcal{Z}(d) \rangle$ where $\mathcal{L}(d)$ is a set of labels and \mathcal{Z} is a subset of $\mathcal{P}(\mathcal{L})$, the power set of $\mathcal{L}(d)$, called the *abstract zone set* of d .

If there is a bijection between the curves of concrete diagram d and the labels of an abstract diagram d' that induces a bijection between the concrete zones of d and the abstract zones of d' , then d is a *realisation* of d' , or d' is the *abstraction* of d .

Wellformedness conditions. Topological and geometric conditions that are placed on concrete Euler diagrams are often called wellformedness conditions. These are typically enforced with the aim of reducing the potential of errors in comprehension of users, especially novices. The set of wellformedness conditions we adopt are:

- WF1 Curves can only intersect transversely; that is curves cannot meet tangentially.
- WF2 Intersection points can only have multiplicity 2; that is at most two curves can intersect at any given point.
- WF3 Zones are connected; that is each zone corresponds to a minimal regions.

These conditions, as used in [19], are commonly enforced, but in some applications individual conditions are relaxed. From left to right, Fig. 3 shows three non-wellformed EDs with: two curves intersecting tangentially; three curves intersecting in a single point of multiplicity 3 (sometimes called a triple point); a disconnected zone, shown shaded.

Now, since we wish to consider applications involving natural operations on wellformed Euler diagrams such as adding and removing curves, and we wish to preserve the wellformedness of the diagrams upon such transformations, it makes sense to restrict to the class of *reducible* Euler diagrams (reducible Venn diagrams are defined in [35]).

Definition 4. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed Euler diagram where $\mathcal{C} = \{C_1, C_2, \dots, C_n\}$. Then d is said to be *reducible* iff there exists a permutation $\pi \in S_n$ such that all of the following n diagrams are wellformed:

$$\begin{aligned}
 d_1 &= \langle \{C_{\pi_1}\}, \mathcal{Z}_1 \rangle, \\
 d_2 &= \langle \{C_{\pi_1}, C_{\pi_2}\}, \mathcal{Z}_2 \rangle, \\
 &\vdots \\
 d_{n-1} &= \langle \{C_{\pi_1}, C_{\pi_2}, \dots, C_{\pi_{n-1}}\}, \mathcal{Z}_{n-1} \rangle, \\
 d_n &= d = \langle \mathcal{C}, \mathcal{Z} \rangle,
 \end{aligned}$$

where π_i denotes $\pi(i) \in \{1, \dots, n\}$.

However, there exist Euler diagrams which are not reducible (i.e. cannot be constructed as a sequence of curve additions keeping a wellformed diagram at each step) but can be constructed via a sequence of additions and removals of curves, always keeping a wellformed diagram at each step. For instance, the example of the symmetric Venn(5) which is drawn with ellipses is irreducible [35] but it can be constructed via a sequence of additions and removals of curves. We will call this broader class of diagrams *weakly reducible* Euler diagrams. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$, then $d + A$ and $d - B$ denotes respectively the *addition of a curve* $A \notin \mathcal{C}$ to d and the *removal of a curve* $B \in \mathcal{C}$.

Definition 5. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed Euler diagram. Then d is said to be *weakly reducible* if there exists a finite sequence of wellformed Euler diagrams $d_0 = \langle \emptyset, \{\emptyset\} \rangle, d_1, \dots, d_{m-1}, d_m = d$ such that for each $i \in \{1, \dots, m\}$, we have either $d_i = d_{i-1} + A_i$ or $d_i = d_{i-1} - A_i$.

So any weakly reducible Euler diagram can be viewed as a sequence of additions of curves, keeping a wellformed diagram at each step.

3. Basic results

In this section we relate properties of the curves (such as their intersections) and properties of the regions bounded by those curves (e.g. the intersections of the interiors of the regions bounded by the curves).

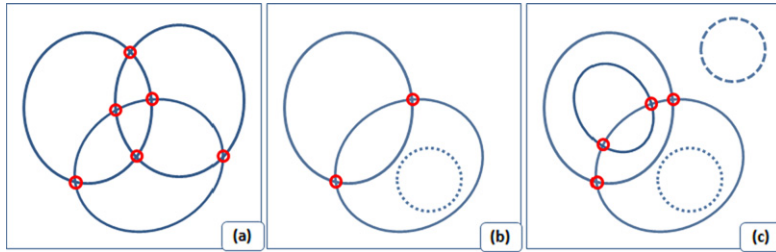


Fig. 4. Single curve components are shown dashed, and intersection points are highlighted: (a) 6 intersection points and 1 connected component; (b) 2 intersection points and 2 connected components; (c) 4 intersection points and 3 connected components.

Definition 6. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a concrete Euler diagram and A a curve. Let

1. $Over(A)$ be the set of curves in \mathcal{C} which properly overlap with A (that is $Over(A) = \{C \in \mathcal{C} \text{ such that } A \cap C \neq \emptyset\}$);
2. $Cont(A)$ be the set of curves in \mathcal{C} which properly contain A (that is $Cont(A) = \{C \in \mathcal{C} \text{ such that } C \notin Over(A) \text{ and } int(A) \cap int(C) = int(A)\}$);
3. $Inter(A)$ be the set of $O(|\mathcal{C}|)$ intersection points between A and any curve in \mathcal{C} .

For instance, in Fig. 2(a) we have $Over(A) = \{D\}$, $Cont(A) = \emptyset$ and $Inter(A)$ contains the two intersection points between A and D .

We assume that, given two curves A and B of an Euler diagram, we can quickly find:

1. the relationships between A and B ; that is whether A and B overlaps or one contains the other;
2. their intersection points if A and B properly overlap;
3. the relationship between any given point $x \in \mathbf{R}^2$ and A ; that is whether x belongs to A , $int(A)$ or $ext(A)$.

For example, in the case that each curve is a simple geometric shape, such as a circle or an ellipse, these computations reduce to solving a system of two quadratic equations (1 and 2) and a quadratic equation (3), which can be computed very quickly (with different methods having different time/precision tradeoffs).

Definition 7. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed Euler diagram. Then an *intersection point* of d is a point of intersection of the curves in \mathcal{C} . The number of intersection points of d is denoted by $ip(d)$. The set of curves in d can be partitioned into the connected components of d (i.e. the maximal sets of intersecting curves). We denote the number of these components as $cc(d)$.

Since one of the goals of our algorithm is to be able to check the wellformedness of a given diagram and its wellformedness after the addition of a new curve or the removal of one of its curves, in the following we present a simple algorithm for checking the wellformedness of a diagram at drawing time (so we can highlight this information for a user, for instance). The preservation of first two wellformed conditions (WF1, WF2) upon curve addition or removal is easily checked: each intersection of the new curve with existing curves should create a transverse crossing intersection point (WF1) which is different from the existing intersection points (WF2). The main difficulty is the recognition of disconnected zones (WF3). We relate the number of zones of a wellformed Euler diagram d to the number of intersection points of d and to the number of connected components of d . Fig. 4 shows three examples of Euler diagrams with different numbers of connected components.

Theorem 1. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be obtained by adding or removing a curve from a weakly reducible Euler diagram. If d satisfies WF1 and WF2 then d is wellformed iff $|\mathcal{Z}| = ip(d) + cc(d) + 1$.

Proof. First, suppose d is connected (so $cc(d) = 1$). If d consists of a single curve then we have 2 regions, $ip(d) = 0$, and so the result holds.

If d consists of more than one curve then we can view d as a graph with vertices at the intersection points (called the graph of the diagram in [3]). Since d satisfies WF2 then each vertex has degree 4. Since every edge is incident with 2 vertices, we have that the number of edges, $e = \frac{4 \cdot ip(d)}{2} = 2 \cdot ip(d)$. Furthermore, by Euler’s formula we have $v - e + f = 2$, where f is the number of faces which corresponds to $|\mathcal{Z}|$ (the number of minimal regions in the diagram) whenever d satisfies WF3, $v = ip(d)$ and so, $f = 2 - ip(d) + 2 \cdot ip(d) = 1 + ip(d) + cc(d)$ as required.

Now suppose that d is disconnected. Then the application of the above argument to each connected component yields the correct result for the individual components. However, when considering the disjoint union of any two non-single curve components we will have double-counted exactly one face. More explicitly, suppose that one such component C' , viewed as a graph, has f' regions and v' vertices, so $f' = 1 + v' + 1$, and component C'' has f'' regions and v'' vertices, so

Algorithm 1: NewCurveZoneComputationTrivial($d, A, \mathcal{X}_{\mathcal{Z}}$)

```

Input: An Euler diagram  $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ ; a curve  $A$  such that  $A \notin \mathcal{C}$ ;  $\mathcal{X}_{\mathcal{Z}}$ , the set of zone descriptors for  $d$ .
Output:  $\mathcal{X}_{\mathcal{Z}'}$ , the set of zone descriptors for  $d' = d \cup A = \langle \mathcal{C} \cup \{A\}, \mathcal{Z}' \rangle$ .

1  $\mathcal{X}_{\mathcal{Z}'} := \mathcal{X}_{\mathcal{Z}}$ ; // The list of zone descriptors of  $d$  which will be updated by comparing with the new curve  $A$ 
2 forall zone descriptors  $X_z$  in  $\mathcal{X}_{\mathcal{Z}}$  do
3    $Temp := X_z \cup \{A\}$ 
4    $\mathcal{X}_{\mathcal{Z}'} := \mathcal{X}_{\mathcal{Z}'} \cup \{Temp\}$ 
5 end
6 forall zone descriptors  $X_z$  in  $\mathcal{X}_{\mathcal{Z}'}$  do
7   Let  $z$  be the set of zones described by  $X_z$ 
8   if CheckZone( $z$ ) = false then // Check whether the zone  $z$ , described by  $X_z$ , is present or not in  $d'$ 
9     remove  $X_z$  from  $\mathcal{X}_{\mathcal{Z}'}$ 
10  end
11 end
12 return  $\mathcal{X}_{\mathcal{Z}'}$ 

```

$f'' = 1 + v'' + 1$. Then $C' \sqcup C''$ has $f' + f'' - 1 = 1 + v' + 1 + 1 + v'' + 1 - 1 = 1 + v' + v'' + 2$ regions, as required. Finally, we note that taking the disjoint union with any single-curve component effectively adds 1 region so that such an addition preserves the required relation. The result follows. \square

4. A naive algorithm for the on-line computation of the zone set

We present a naive algorithm that provides a solution to the problem of computing the zone set in an on-line fashion. It will be used as a baseline for comparison with the efficient algorithm introduced in Section 5. In the following we use the notation ‘:=’ to mean set assignment (i.e. $X := Y$ means assign the set Y to the variable X).

The naive algorithm (Algorithm 1) implements the following idea.⁴ When a new curve A is added to a given diagram $d = \langle \mathcal{C}, \mathcal{Z} \rangle$, for which we know the zone descriptors, it prepares a list of new candidate zone descriptors for d' (by duplicating the set of zone descriptors $\mathcal{X}_{\mathcal{Z}}$ and adding A to one half of them), which is d with A added, and then for each of these it checks whether the described zone is actually present in d' . The check on the existence of a zone is achieved by the function **CheckZone** which depends on the properties of the curves that are present in the diagram. For example, if all of the curves are circles or ellipses then every zone in the diagram corresponds to the solution of a specific system of $|\mathcal{C}|$ inequalities, and every new curve just adds another equation to the system. It is worth noting that in case the sets are defined by using polygons, that is connected segments, the intersections can be computed by using one of the methods presented in [33]. Even in this less general scenario, the time complexity to compute the intersections is, at least, linear in the number of segments used to represent all the sets in the diagram.

For all the cases above, the function **CheckZone** requires a significant amount of time, clearly non-constant with respect to the number of curve present in d . Let $f(|\mathcal{C}|)$ denote the time taken by the function **CheckZone**. Then the complexity of the naive algorithm is easily calculated as $O(f(|\mathcal{C}|) \cdot |\mathcal{Z}|)$.

5. An efficient zones computation strategy

In this section we present a novel methodology, which efficiently computes the zones present in a weakly reducible wellformed diagram. We aim to solve the following problems: given a wellformed Euler diagram $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ with $A \in \mathcal{C}$ and $B \notin \mathcal{C}$ such that $d + A$ and $d - B$ are wellformed, compute the collection of zone descriptors for $d + A$ and $d - B$. Indeed by solving such problems we will be able to find the collection of abstract zones associated to each concrete weakly reducible Euler diagram.

Definition 8. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed concrete Euler diagram and let $A \notin \mathcal{C}$ a curve, then:

- each zone $z \in \mathcal{Z}$ that is completely covered by A , i.e. $z \subset \text{int}(A)$, is called a *covered zone*;
- each zone that is partially covered by A (i.e. z intersects both $\text{int}(A)$ and $\text{ext}(A)$ non-trivially) is called a *split zone*.

In this last case we say that z is split by A into two zones $z' = z \cap \text{int}(A)$ and $z'' = z \cap \text{ext}(A)$. Let $Z_c \subseteq \mathcal{Z}$ denote the set of covered zone, let $Z_s \subseteq \mathcal{Z}$ denote the set of split zones and let $Z_n \subseteq \mathcal{Z}$ denote the set of new zones (the portions of the split zones that are covered by A , i.e., the zones z' above).

We denote with \mathcal{X}_{Z_c} , \mathcal{X}_{Z_s} and \mathcal{X}_{Z_n} respectively the set of covered zones, split zones and new zones zone descriptors.

⁴ The most naive version would in fact just check for the presence of all 2^n zones but this is unnecessarily bad.

5.1. The key ideas of the approach

The key observation is that for curve additions and removals that preserve the wellformedness conditions the effect on the abstract diagram can be easily determined by computing the set of zones that are *split* by A (i.e., Z_s) and the set of zones that are *covered* by A (i.e., Z_c); that is this enables the computation of set of zone descriptors for the resulting diagram d' . In particular:

- By analysing the intersection points generated by A (i.e., points in $Inter(A)$) one can identify Z_s and the corresponding Z_n ;
- Moreover, each zone of d can be represented using a single marked point, suitably chosen. These points and their associations to zones can be quickly updated upon the addition or removal of curves, and so one can use them to quickly compute Z_c .

We introduce in [6] the new concept of marked Euler Diagrams, which are Euler diagrams together with a distinguished set of points which mark the zones of the diagram. This enables the tracking of the zone set via this set of marked points and is a fundamental concept utilised in our zone computation algorithm.

Definition 9. Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a weakly reducible Euler diagram. We say d is a *marked Euler diagram* if there is an injective function $m : \mathcal{Z} \rightarrow \mathbb{R}^2$, called a *marking function*, such that:

1. if z is the *Universe* (that is $X_z = \{\emptyset\}$) then $m(z) \in \bigcap_{c_j \in \mathcal{C}} ext(c_j)$ ⁵;
2. otherwise $m(z) \in cl(z)$, the closure in \mathbb{R}^2 of the open region z .

For each zone $z \in \mathcal{Z}$, the image $m(z)$ ⁶ is called a *marked point*.

In Figs. 5, 6 and 7 all of the arrowed-dots indicate marked points. The arrow indicates the zone marked by the marked point. The idea is that we have a unique marked point associated with each zone of the diagram. For all zones, except the outside zone, these marked points are points on the curves (the boundaries of the zones). The marked points are used to quickly check whether a zone which is not split by a new curve A is covered by A (in time constant with respect to the number of curves present in the diagram).

A side effect of our strategy is that we need to update the marked points and their associations to zones upon the addition or the removal of curves so that we retain a marked Euler diagram. By Theorem 1 we know that the number of marked points required for a given weakly reducible diagram is equal to the number of intersection points of d plus the number of connected components plus the marked point for the Universe. In particular, using our strategy we will have that for each connected component d_i , for $i = 1, \dots, cc(d)$, of a given diagram d , its zones are marked by the intersection points of d_i plus one point which belongs to the boundary of one of the curves in d_i .

The idea will be to consider any weakly reducible diagram as a sequence of additions/removals of curves. For each curve added which forms a new component we can add a marked point anywhere on the curve, and for each curve added that does not form a new component we can use the points generated by the new curve as marked points. Theorem 1 tell us that we have enough marked points to record the zone set. The clever management of these marked points is what enables an efficient algorithm to be produced.

Let us consider now the addition of a new curve to a given diagram (the following arguments apply also for curve removal). The method we use involves updating the zone set descriptors upon the addition of new curves to the diagram, and hence upon the updating of the marked points. On the left of Fig. 5(a) we have a zone z (bounded by a curve C in this instance) having marked point $m(z)$. On the right-hand side of Fig. 5(a) we have the same diagram after the addition of another curve A , which has split the zone z into two zones, depicted as $z' = z \cap int(A)$ and $z'' = z \cap ext(A)$. It also shows the update of the marked points upon the addition of A . Notice that z and z'' have the same zone descriptor, while the zone descriptor for z' is easily obtained by adding A to the zone descriptor of z (i.e., $X_{z''} = X_z$ and $X_{z'} = X_z \cup \{A\}$). For this reason we will refer to z' as the new zone generated by A (that is $z' \in Z_n$) and to z'' as the old zone. With reference to the figure, we will use the following method to automatically update marked points after the addition of a new curve A that intersects $\partial(z)$, where x is an intersection point between A and $\partial(z)$ (the boundary of z):

1. If $m(z) \in int(A)$, then $m(z)$ is assigned to z' and x is assigned to z'' , as shown in Fig. 5(a). That is, the marked point for zone z before the addition becomes the marked point for zone z' after the addition of A and a new marked point x , generated by A , is used to mark zone z'' . We will refer to this operation as a *swapping*; an existing marked point is reassigned to a newly created zone while the old zone is marked by a new point.

⁵ This part could be removed if we formalised the diagrams as containing a bounding box as a curve and then the marked point for the outside zone would be on this bounding box.

⁶ Sometimes we refer to $m(z)$ as x_z .point to emphasise the fact that it is a point that marks a zone z described by X_z .

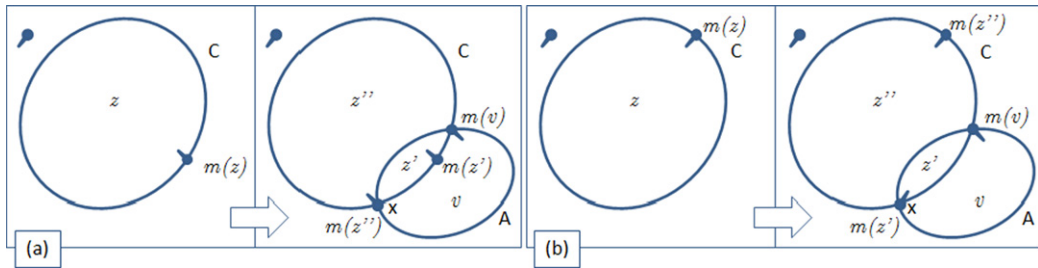


Fig. 5. Updating the set of marked points for the zones upon the addition of a new curve A for two cases: (a) the marked point is in $\text{int}(A)$; (b) the marked point is not in $\text{int}(A)$.

2. If $m(z) \notin \text{int}(A)$, then $m(z)$ is assigned to z'' and x is assigned to z' , as shown in Fig. 5(b). That is, the marked point for zone z before the addition remains unchanged (that is, it now marks z''), and the point x generated by A is used as the marked point for the new zone z' .

Analogously our algorithm performs one of the two step above for each other intersection point in $\text{Inter}(A)$. In this particular case the remaining new zone v having zone descriptor $\{A\}$ is marked by the remaining intersection point between A and the boundary of z , dubbed $m(v)$.

Remark 1. When viewing a diagram d as a sequence of curve additions, the choice of ordering of the curves can give rise to different marked point associations. Anyway, this does not preclude either the correctness or the efficiency of our algorithms.

Given a wellformed reducible diagram $d = \langle C, \mathcal{Z} \rangle$ and a curve A such that $A \notin C$ (resp. B such that $B \in C$) and $d' = d + A$ (resp. $d' = d - A$) is wellformed, Algorithm **NewCurveZoneComputation** (resp. **DeleteCurveZoneComputation**), analyses the relationships between the curve A (resp. B), the curves in C and the marked points associated to the zones in \mathcal{Z} , in order to compute the new collection of zone descriptors associated to d' . This algorithm also updates the marked points set consistently. These results are synthesized by the following theorem:

Theorem 2. Let $d = \langle C, \mathcal{Z} \rangle$ be a wellformed Euler diagram. Then

- (i) If $A \notin C$ and $d + A$ is wellformed, then the procedure **NewCurveZoneComputation**(d, A) computes the new collection of zone descriptors for the zones \mathcal{Z}' of $d' = \langle C \cup A, \mathcal{Z}' \rangle$.
- (ii) If $B \in C$ and $d - B$ is wellformed, then the procedure **DeleteCurveZoneComputation**(d, B) computes the new collection of zone descriptors for the zones \mathcal{Z}' of $d' = \langle C - B, \mathcal{Z}' \rangle$.

Moreover, both procedures compute, for each zone $z \in \mathcal{Z}' - \{\text{outer zone}\}$, the marked point $m(z)$ such that $m(z)$ belongs to $\text{cl}(z)$, and they have running time $O(|\mathcal{Z}'| + |C| \log |C|)$.

Remark 2. We only give details of the case when a new curve is added to an Euler diagram d , showing the strategy which provides the sets Z_s and Z_c . However, the ideas easily extend to the case of removal of a curve A which belongs to d (using the sets Z_s and Z_c one can easily update the collection of zone descriptors associated to the resulting diagram $d - \{A\}$).

5.2. Some illustrative examples

Before showing the formal description of our solution, we describe it informally, through some examples.

The first example, depicted in Fig. 6, shows a Venn4 diagram generated through a sequence of four curve additions, starting from an empty diagram. In the following, we refer to diagrams d_1, d_2, d_3 and d_4 for the diagrams with 1, 2, 3 and 4 curves respectively, depicted in the Fig. 6. When the first curve A is added (cf. Fig. 6(a)) there are no intersection points. In this case the new curve generates exactly one new zone (described by $\{A\}$), which is obtained splitting the Universe. Specifically, d_1 has 1 curve A and 2 zones described by \emptyset and $\{A\}$. The new zone's marked point is chosen as an arbitrary point on the curve A . The second curve B intersects the diagram d_1 in two points, x_0 and x_1 and generates two new zones (cf. Fig. 6(b)). The new zones are obtained by splitting both the zones described by $\{A\}$ and \emptyset (the Universe) in d_1 (that is, $\mathcal{X}_{Z_s} = \{\{A\}, \emptyset\}$ and $\mathcal{X}_{Z_n} = \{\{A, B\}, \{B\}\}$ where \mathcal{X}_{Z_s} and \mathcal{X}_{Z_n} respectively represent the collection of zone descriptors of the split zones and the collection of zone descriptors of the new zones). Specifically, d_2 has 2 curves A and B and 4 zones described by $\emptyset, \{A\}, \{B\}$ and $\{A, B\}$. Marked points associated to the new zones are easily computed using the method described in Section 5.1; in this particular case they correspond with the intersection points, x_0 and x_1 between A and B . Either of the two intersection points x_0 and x_1 can mark either of the zones described respectively by $\{A, B\}$,

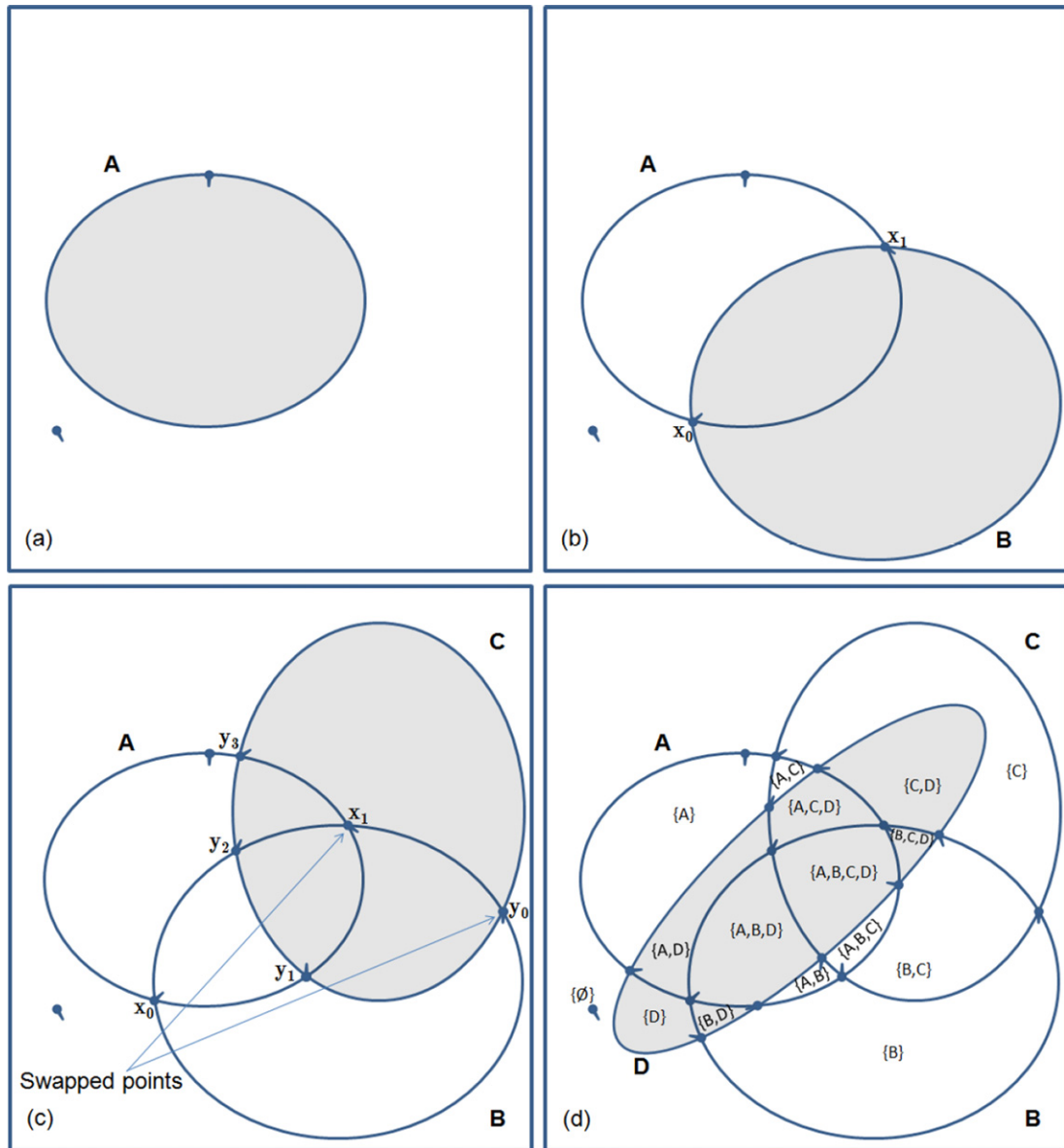


Fig. 6. A sequence of curve additions with marked points associations; For each step the added curve is showed shaded: (a) A curve A is added to an empty diagram, the marked point for the curve A is chosen arbitrarily on the curve; (b) A new curve B is added generating two intersection points x_0, x_1 and two new zones. Each intersection point is then used as marked point for a new zone; (c) A new curve C is added generating four intersection points y_0, y_1, y_2, y_3 and four new zones. In this case the marked points association has been obtained using also the *swapping* operation (cf. Section 5.1); (d) A new curve D is added generating a Venn4 diagram.

$\{B\}$ but x_0 is chosen to mark $\{A, B\}$ and x_1 is chosen to mark $\{B\}$ here. The third curve C intersects the diagram d_2 in four points, y_0, y_1, y_2 and y_3 , and generates four new zones (cf. Fig. 6(c)). In this case $\mathcal{X}_{Z_s} = \{\emptyset, \{A\}, \{A, B\}, \{B\}\}$ and $\mathcal{X}_{Z_n} = \{\{C\}, \{A, C\}, \{A, B, C\}, \{B, C\}\}$. Moreover, applying the method described in Section 5.1 we remark that in this case a swapping operation is performed. In particular, the marked point x_1 , previously assigned to an old zone (described by $\{B\}$ in d_2), is assigned to a new zone (described by $\{B, C\}$ in d_3) and accordingly the intersection point y_0 , between C and d_2 , is assigned to the zone previously marked with x_1 . Finally, the Venn4 diagram (cf. Fig. 6(d)) is obtained by adding to d_3 a curve which generates exactly 8 new zones and consequently 8 intersection points.

The second example, depicted in Fig. 7, shows three significant cases. The starting diagram d'_4 is obtained adding a new curve D to the diagram d_3 in Fig. 6(c) (cf. Fig. 7(a)). The first case depicted in Fig. 7(b) shows what happens when a new curve E , which does not intersect any other curve in d'_4 is added generating d'_5 . The curve E belongs to a new connected component, splits the zone in d'_4 described by the set $Cont(E) = \{C, D\}$ and generates a new zone, described by the set $Cont(E) \cup \{E\}$ in d'_5 . In a manner similar to that of the first curve of a diagram, the new zone marked point is chosen as

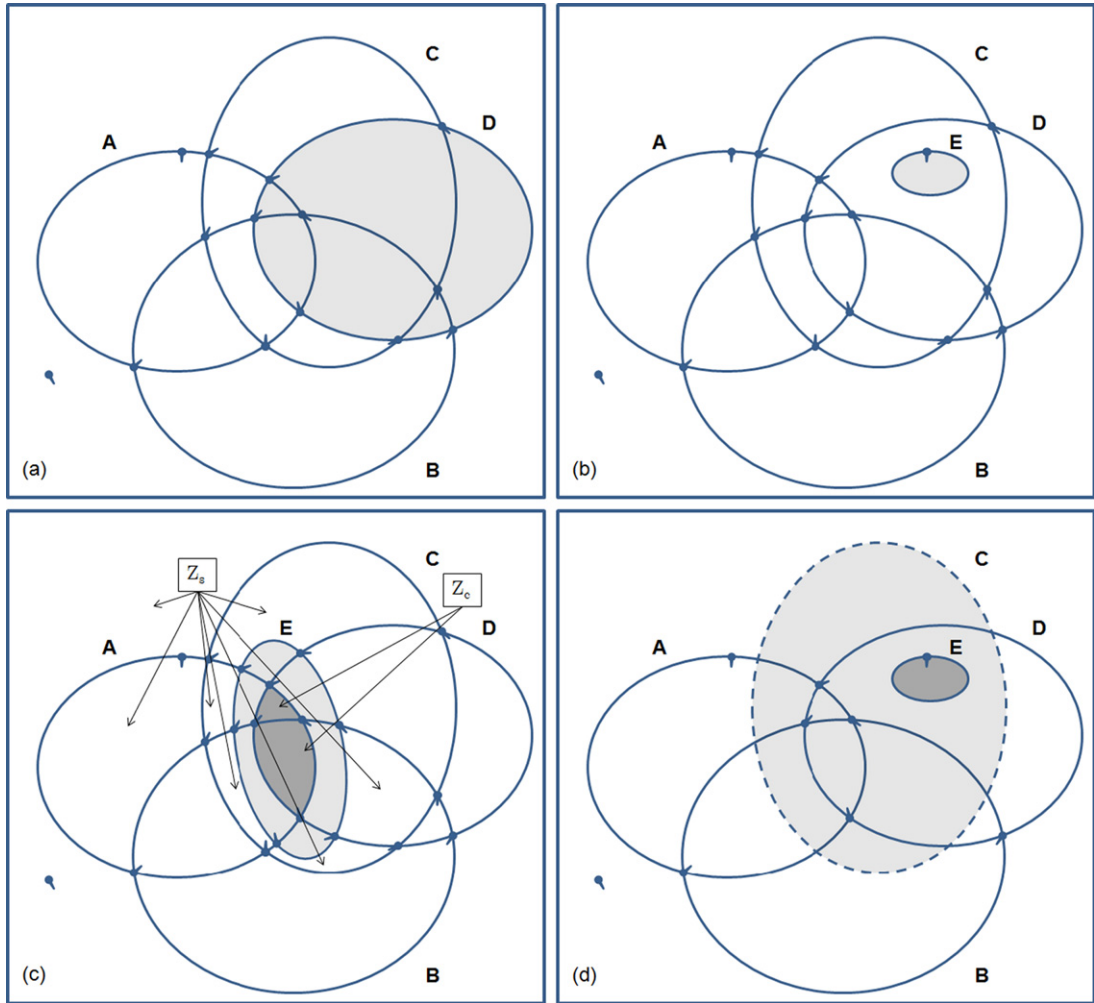


Fig. 7. (a) The diagram d'_4 is obtained adding a new curve D to the diagram in Fig. 6(c). (b) The diagram d'_5 : The curve E does not intersect any other curve of d'_4 ; the zone described by $\{C, D\}$ is split into two zones in d'_5 . (c) The diagram d''_5 : The curve E intersects several curves of d'_4 ; the set Z_s of split zones and the set Z_c of zones covered by E in d'_4 need to be identified. (d) An example of curve removal: The curve E is removed from d'_5 .

an arbitrary point on the curve E . Alternatively, if the curve E intersects the diagram d'_4 in several points generating d''_5 (cf. Fig. 7(c)). This case shows that not only do we need to identify the set of split zones Z_s and the corresponding set of new zones Z_n , but we also need to identify the set of zones Z_c which are in the interior of E (i.e. which are covered by E). In this case we have $\mathcal{X}_{Z_c} = \{\{A, B, C, D\}, \{A, C, D\}\}$ where \mathcal{X}_{Z_c} denotes the collection of zone descriptors of the covered zones. These zones can be identified using their marked points. Finally, Fig. 7(d) depicts an example of curve removal. Specifically the curve C is removed from the diagram d'_5 . This example shows that the removal operation can be easily obtained by considering the curve to be removed as the last curve added to the diagram. Hence we can perform a curve removal by doing the same operation for a curve addition but in reverse order (rollback).

5.3. Formal description and proofs

In this section we provide the formal algorithms and complexity analysis. In order to provide a clear exposition, we will present Algorithm 3, called **NewCurveZoneComputation** as a sequence of 4 steps: *Compute A's relationship with d*; *Compute split zones*; *Updated marked points*; *Compute covered zones*.

5.3.1. Compute A's relationship with d

Algorithm 2, called **ComputeCurveRelationships**, computes the relationship between each curve $B \in C$ and the new curve A to be added to d . Using Definition 6, one can easily prove the following lemma.

Lemma 1. Given a collection of curves C and a curve A the procedure **ComputeCurveRelationships** computes $\text{Cont}(A)$, $\text{Over}(A)$ and $\text{Inter}(A)$. Moreover, for each point in $x \in \text{Inter}(A)$, x keeps a reference to the curve which, intersecting A , generates it.

Algorithm 2: ComputeCurveRelationships(d, A)**Input:** A wellformed Euler diagram $d = \langle C, \mathcal{Z} \rangle$; a curve A not present in C .**Output:** Computes the sets $Cont(A)$, $Over(A)$ and $Inter(A)$. Each point in $Inter(A)$ keeps a reference to the curve which, intersecting A , generates it.

```

1  $Cont(A) := Over(A) := Inter(A) := \emptyset$ 
2 forall  $B \in C$  do
3   if  $B$  properly contains  $A$  then
4      $Cont(A) := Cont(A) \cup \{B\}$ 
5   else
6     if  $A$  and  $B$  properly overlap then
7        $Over(A) := Over(A) \cup \{B\}$ 
8       let  $X_B = \{x_0, x_1, \dots, x_{m-1}\}$  be the set of intersection points between curves  $A$  and  $B$ 
9       forall  $x \in X_B$  do
10         $x.genCurve := B$ ; // each point keeps a reference to the curve which, intersecting  $A$ , generates it
11         $Inter(A) := Inter(A) \cup \{x\}$ 
12      end
13    end
14  end
15 end
16 return ( $Cont(A), Over(A), Inter(A)$ )

```

5.3.2. Compute split zones

This phase, described by lines 2–10 of Algorithm **NewCurveZoneComputation**, aims to compute the set of zones that are split by A (i.e., Z_s) and the corresponding set of zones that are covered by A (i.e., Z_c). There are two cases (i) if $Over(A) = \emptyset$, and (ii) if it is not empty.

Case (i): There are no intersections created by the addition of A so A belongs to a new connected component. Since A does not intersect with any other curve of C , A splits only the zone z having zone descriptor $X_z = Cont(A)$; see curve E in Fig. 7(b). Hence, a new zone z' , having zone descriptor $X_{z'} = Cont(A) \cup \{A\}$, is added to Z_n (the collection of new zones), which represents the region of z which is in the interior of A . Furthermore the zone $z'' = z \cap int(A)$ (having zone descriptor $X_{z''} = X_z = Cont(A)$) is added to Z_s (the collection of split zones). The new zone's marked point $m(z') = X_{z'}.point$ is an arbitrarily chosen point in A (lines 3–7). Notice that when A is not contained in any $B \in C$ (that is, $Cont(A) = \emptyset$), the split zone z is the *Universe* and the new zone z' has zone descriptor $\{A\}$.

Case (ii): The curve A splits several zones (see Fig. 7(c)). In this case Algorithm 4, called **ComputeSplitZones** computes \mathcal{X}_{Z_s} , the collection of zone descriptors of the split zones and the corresponding \mathcal{X}_{Z_n} , the collection of zone descriptors of the new zones. The algorithm **ComputeSplitZones** exploits the following observation in order to quickly compute the zone descriptors for the zones split by A .

Observation 1. Let $\{x_0, x_1, \dots, x_{m-1}\}$ be all of the intersection points that we meet as we traverse the curve A from an arbitrary point on A .

- (i) For each $i = 0, \dots, m - 1$ each arc $(x_i, x_{i+1 \bmod m})$ splits exactly one zone (note that two arcs can split the same zone but one arc cannot split more than one zone) of d .
- (ii) Moreover two consecutive arcs $(x_i, x_{i+1 \bmod m})$ and $(x_{i+1 \bmod m}, x_{i+2 \bmod m})$ split two zones such that their zone descriptors differ by exactly one curve (the curve that intersects with A generating the intersection point $x_{i+1 \bmod m}$).

The points in $Inter(A)$ partition the curve A into a set of m arcs. The rationale is to determine all of the zones split by one such arc. The arcs are analysed in the sequence they are met as one traverses the curve (line 1). Formally, let $\{x_0, x_1, \dots, x_{m-1}\}$ be all of the intersection points that we meet as we traverse the curve A clockwise from an arbitrary point on A . For each $i = 0, \dots, m - 1$ each arc $(x_i, x_{i+1 \bmod m})$ splits exactly one zone. The procedure computes for each arc $(x_i, x_{i+1 \bmod m})$ the set $x_i.zone$ which corresponds to the zone descriptor of the zone split by the arc $(x_i, x_{i+1 \bmod m})$.

Let x_{01} be an arbitrary point on the arc (x_0, x_1) . The algorithm computes the zone descriptor $x_{01}.zone$ of the zone split by the arc (x_0, x_1) by checking the set of curves which x_{01} belongs to. First, since, by definition, each curve $C \in Cont(A)$ properly contains A and therefore also x_{01} , all the curves in $Cont(A)$ are added to $x_{01}.zone$. Then, for each curve $B \in Over(A)$, the algorithm checks whether $x_{01} \in int(B)$ or not (line 4–8) and accordingly updates $x_{01}.zone$.

Each successive zone descriptor is calculated (cf. Observation 1) by computing the difference with respect to the previously computed zone. For each $i = 1, \dots, m - 1$, the zone descriptor $x_i.zone$ of the zone split by the arc (x_i, x_{i+1}) is computed using the set $x_{i-1}.zone$ and the curve G which, intersecting A , has generated the point x_i . That is, if G belongs to $x_{i-1}.zone$, then $x_i.zone$ is obtained by removing G from $x_{i-1}.zone$, otherwise $x_i.zone$ is obtained by adding G to $x_{i-1}.zone$ (lines 9–16).

Remark 3. Notice that we could also compute the zone split by each arc $(x_i, x_{i+1 \bmod m})$ using the same strategy adopted for the arc (x_0, x_1) . In that case we do not need to order the points in $Inter(A)$. On the other hand, using this naive approach

Algorithm 3: NewCurveZoneComputation($d, A, \mathcal{X}_{\mathcal{Z}}$)

Input: A wellformed Euler diagram $d = \langle C, \mathcal{Z} \rangle$; a curve A such that $A \notin C$ and $d \cup A$ is wellformed; $\mathcal{X}_{\mathcal{Z}}$, the set of zone descriptors for d .
Output: $\mathcal{X}_{\mathcal{Z}'}$, the collection of zone descriptors of $d' = \langle C \cup \{A\}, \mathcal{Z}' \rangle$.

```

1  ( $Cont(A), Over(A), Inter(A)$ ) := ComputeCurveRelationships( $d, A$ )
2  if  $Over(A) = \emptyset$  then
3       $X_z := Cont(A)$ ;
4       $X_{z'} := Cont(A) \cup \{A\}$ ;
5       $X_z.point :=$  an arbitrary point on  $A$ 
6       $\mathcal{X}_{Z_s} := \mathcal{X}_{Z_s} \cup \{X_z\}$ ;
7       $\mathcal{X}_{Z_n} := \mathcal{X}_{Z_n} \cup \{X_{z'}\}$ ;
8  else
9      ( $\mathcal{X}_{Z_s}, \mathcal{X}_{Z_n}$ ) := ComputeSplitZones( $A, Cont(A), Over(A), Inter(A)$ )
10 end
11 if  $d + A$  is wellformed then
12     forall  $X_z \in \mathcal{X}_{\mathcal{Z}} - \mathcal{X}_{Z_s}$  do
13         if  $X_z.point \in int(A)$  then
14              $X_z := X_z \cup \{A\}$ ;
15         end
16     end
17      $\mathcal{X}_{Z'} := \mathcal{X}_{\mathcal{Z}} \cup \mathcal{X}_{Z_n}$ ;
18     return ( $\mathcal{X}_{Z'}$ )
19 else
20     return ( $\mathcal{X}_{\mathcal{Z}}$ );
21 end

```

// A does not properly overlap any curve present in C
// z is the split zone described by the curves in $Cont(A)$
// z' is the new zone generated by A
// \mathcal{X}_{Z_s} is the collection of zone descriptors of the split zones
// \mathcal{X}_{Z_n} is the collection of zone descriptors of the new zones
// wellformedness check
// for each non-split zone
// if z is covered by A ; $z \in Z_c$
// X_z is updated
// $\mathcal{X}_{Z'}$ contains the collection of zone descriptors of d'
// The curve A is discarded

Algorithm 4: ComputeSplitZones($A, Cont(A), Over(A), Inter(A)$)

Input: The new curve A ; the sets $Cont(A)$, $Over(A)$ and $Inter(A)$.
Output: \mathcal{X}_{Z_s} and \mathcal{X}_{Z_n} , the collection of zone descriptors of the zones split by A and the corresponding set of new zones.

```

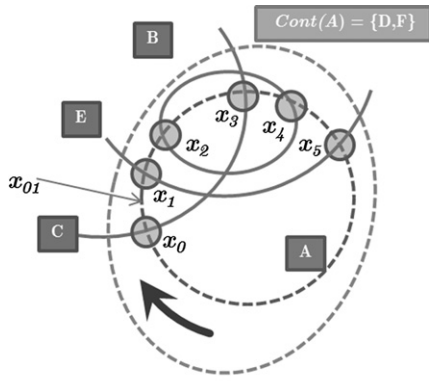
1  Sort points in  $Inter(A)$  along the curve and let  $(x_0, x_1, \dots, x_{m-1})$  be the sorting
2   $x_0.zone := Cont(A)$ 
3   $x_{01} :=$  any point on the arc  $(x_0, x_1)$ 
4  forall  $B \in Over(A)$  do
5      if  $x_{01} \in int(B)$  then
6           $x_0.zone := x_0.zone \cup \{B\}$ 
7      end
8  end
9  forall  $i = 1, 2, \dots, m-1$  do
10      $G := x_i.genCurve$ ;
11     if  $C \in x_{i-1}.zone$  then
12          $x_i.zone := x_{i-1}.zone - \{G\}$ 
13     else
14          $x_i.zone := x_{i-1}.zone \cup \{G\}$ 
15     end
16 end
17 forall  $x \in Inter(A)$  do
18      $X_z := x.zone$ ;
19      $X_{z'} := x.zone \cup \{A\}$ ;
20     if  $X_z.point \in int(A)$  then
21          $X_{z'.point} := X_z.point$ 
22     else
23          $X_{z'.point} := x$ 
24     end
25      $\mathcal{X}_{Z_s} := \mathcal{X}_{Z_s} \cup \{X_z\}$ ;
26      $\mathcal{X}_{Z_n} := \mathcal{X}_{Z_n} \cup \{X_{z'}\}$ ;
27 end
28 return ( $\mathcal{X}_{Z_s}, \mathcal{X}_{Z_n}$ )

```

// Computing the zone descriptor for the zone split by (x_0, x_1)
// Computing the zone descriptor for the zones split by the arcs $(x_1, x_2), \dots, (x_{m-1}, x_0)$
// G is the collection of curves which pass through x_i
// z is split by A
// z' is the new zone
// the points need to be swapped
// \mathcal{X}_{Z_s} is the collection of zone descriptors of the split zones
// \mathcal{X}_{Z_n} is the collection of zone descriptors of the new zones

the complexity of lines (9–16) become larger. Assuming that we are able to check if a point x is inside a generic curve A , in constant time and that we use an efficient data structure for set representation, our approach requires only $O(|C| \log |C|)$ steps to compute the zone descriptors of the zones split by arcs $(x_i, x_{i+1 \bmod m})$ for $i = 0, 1, \dots, m-1$.

Fig. 8 shows a schematic example where the curve A has been added to d creating 6 new intersection points (shown with circles). Arbitrarily choosing the bottom left of these as x_0 , we compute $x_0.zone$ as $\{C, D, F\}$, where the curves D and F properly contain A and therefore x_{01} and the curve C contains x_{01} . In the table of the figure the ' $>$ ' and ' $<$ ' symbols indicate



Intersection points	$x_i.genCurve$	$Cont(A) = \{D,F\}$	B	E	C	$x_i.zone$
x_0	C	x			>	{C,D,F}
x_1	E	x		>	x	{C,D,E,F}
x_2	B	x	>	x	x	{B,C,D,E,F}
x_3	C	x	x	x	<	{B,D,E,F}
x_4	B	x	<	x		{D,E,F}
x_5	E	x		<		{D,F}

Fig. 8. A schematic diagram together with the addition of curve A, which generates 6 new intersection points, shown with circles. Picking an arbitrary starting point on A we traverse A clockwise and order the generated points accordingly. Checking each generated point against the curves that contain or intersect A determines how we update the markings of the diagram.

the curve that is involved in the intersection with the new curve A (entering or leaving the region bounded by the curve, respectively). For instance, since E (the second curve that is met whilst traversing the curve A) is not in $x_0.zone$, we compute $x_1.zone$ as $x_0.zone \cup \{E\}$. On the other hand, when we compute $x_3.zone$, since C is already in $x_2.zone$, $x_3.zone$ is obtained by removing C (i.e. $x_3.zone := x_2.zone - \{C\}$). The zone descriptors of the other zones (i.e., $x_i.zone$ for $i = 1, \dots, m - 1$) are computed similarly.

Lemma 2. The zone descriptors of the zones associated to each point $x \in Inter(A)$ corresponds to the set of zones split by A.

Proof. According to Observation 1, each arc $(x_i, x_{i+1 \bmod m})$ splits one zone.

We will show that for each $i = 0, \dots, m - 1$, the zone split by the arc $(x_i, x_{i+1 \bmod m})$ has zone descriptor $x_i.zone$. By induction on i :

Base ($i = 0$). By construction the set $x_0.zone$ is the zone descriptor of the zone split by the arc (x_0, x_1) .

Inductive step. Assume that the zone split by (x_{i-1}, x_i) has zone descriptor $x_{i-1}.zone$. Since the diagram d is wellformed we have that two consecutive arcs, (x_{i-1}, x_i) and $(x_i, x_{i+1 \bmod m})$, are separated by exactly one curve G. Hence, the zone descriptors of curves which describe two contiguous zones differs by exactly the curve G (i.e., the curve G which A intersects generating the point x_i). Hence, the set $x_i.zone$, which describes the zone split by the arc $(x_i, x_{i+1 \bmod m})$, is easily obtained by adding or removing the curve G from $x_{i-1}.zone$. □

5.3.3. Update marked point

The second part of the procedure **ComputeSplitZones** (lines 17–28) is devoted to updating the marked points for the split zones and the corresponding new zones. For each point $x \in Inter(A)$ the algorithm computes the set X_z which describes a candidate new zone obtained by splitting a zone z described by $X_z = x.zone$. Lines (20–25) compute $X_{z'}.point$ and $X_z.point$. In particular, as described in Section 5.1 (cf. Fig. 5), if the marked point belongs to A (that is if $X_z.point \in int(A)$) then a *swapping* operation is performed: $X_z.point$ is assigned to the zone z' (i.e. $X_{z'}.point := X_z.point$) and x (one of the two intersection point the curve A with the boundary of the zone z) is assigned to z (i.e. $X_z.point := x$); otherwise $X_z.point$ remains assigned to z and x is assigned to z' (i.e. $X_{z'}.point := x$).

Lemma 3. Procedure **ComputeSplitZones** correctly maintains the invariant that the marked points associated to the zones in Z_n and Z_s (except for the outside zone) belong to the closure of the associated zone.

Proof. By Lemma 2 we have that for each $x \in Inter(A)$ a new zone z' , generated by A, is obtained by splitting a zone $z \in Z$ described by $X_z = x.zone$. Indeed, z is divided into two zones: $z' = z \cap int(A)$ and $z'' = z \cap ext(A)$. Here $X_{z'} = x.zone \cup \{A\}$ and $X_{z''} = X_z = x.zone$. We have two cases to consider:

Case 1. If the marked point associated to the split zone z belong to $int(A)$ (i.e., $X_z.point \in int(A)$) then that point is assigned to the new zone z' (i.e., $X_{z'}.point := X_z.point$). Since $X_z.point$ belongs to $cl(z)$, it must belong to $cl(z')$. Then we assign the point x (which is an intersection point between A and $cl(z)$) to the zone z'' having zone descriptor X_z (i.e., $X_{z''}.point := x$). Trivially, $x \in cl(z'')$.

Case 2. Otherwise, the marked point associated to z belongs to $\text{ext}(A)$, and we need only to assign the point x to the new zone z' (i.e. $X_{z'}.point := x$). Trivially, $x \in \text{cl}(z')$. \square

Finally the Algorithm **ComputeSplitZones** returns the two sets \mathcal{X}_{Z_n} and \mathcal{X}_{Z_s} .

5.3.4. Compute covered zones

This last phase, described by lines 12–15 of Algorithm **NewCurveZoneComputation**, aims to compute the set of zones that are covered by A , updating the identifier of each covered zone. Eventually, line 17, updates the collection of zone descriptors $\mathcal{X}_{z'}$ for d' , adding the zones description kept in the collection \mathcal{X}_{Z_n} (the new zones).

Proof of Theorem 2. (i) According to Lemma 2 the Algorithm **ComputeSplitZones** correctly computes the set of split zones. Moreover, lines 12–17 of Algorithm **NewCurveZoneComputation** computes the covered zones. In more detail, for each non-split zone $z \in \mathcal{Z} - Z_s$, the algorithm checks whether z is a covered zone by checking where the marking point $X_z.point$ lies. If $X_z.point \in \text{int}(A)$ then $z \in Z_c$ and z 's zone descriptor is updated to reflect this condition (line 14). Since z is not split by A and the point associated to z lies within A it can be inferred that the whole zone z lies within A . Alternatively, if the point $X_z.point \notin \text{int}(A)$ then the whole zone z lies outside A and z 's zone descriptor does not need to be updated. Hence, the Algorithm **NewCurveZoneComputation** computes the collection of zone descriptors associated to $d' = d + A$.

(ii) By Lemma 3, the procedure **ComputeSplitZones** correctly maintains the invariant that the marked points associated to the zones in Z_n and Z_s (except for the outside zone) belong to the closure of the associated zone. Moreover the marked points associated to non-split zones are not modified by Algorithms 1–4 and therefore they remain valid.

Timing. The invocation of procedure **ComputeCurveRelationships** analyses the relationship between A and each curve in \mathcal{C} , and so takes time $O(|\mathcal{C}|)$. Then, if there are no intersection points generated by A (i.e., if $\text{Over}(A) = \emptyset$) both of the sets \mathcal{X}_{Z_n} and \mathcal{X}_{Z_s} are computed within $O(1)$ time. Instead, if the curve A generates some intersection points, then the procedure **ComputeSplitZones** (lines 1–16) computes the sets \mathcal{X}_{Z_n} and \mathcal{X}_{Z_s} within time $O(|\mathcal{C}| \cdot \log |\mathcal{C}|)$. Indeed, we need $O(|\mathcal{C}| \cdot \log |\mathcal{C}|)$ steps to order the points intersection points generated by the curve A , $O(|\mathcal{C}|)$ step to compute the zone associated to x_0 and $O(|\mathcal{C}| \cdot \log |\mathcal{C}|)$ to compute the zones associated to x_1, x_2, \dots, x_{m-1} . In particular, for each point in $\{x_1, x_2, \dots, x_{m-1}\}$ we need $O(\log |\mathcal{C}|)$ steps to check whether the generating curve G belongs to the previous zone or not.

Finally the covered zones are computed within time $O(|\mathcal{Z}|)$ by analysing the marked point of each non-split zone. Collectively, then, algorithm **NewCurveZoneComputation** operates within time $O(|\mathcal{Z}| + |\mathcal{C}| \log |\mathcal{C}|)$. \square

6. Conclusion

Whilst work on the Euler diagram generation problem (the generation of concrete diagrams from an abstract diagram or description) has been attempted in the past [3,19,20,42] little or no work has been performed on the other direction. This may be since for a human the problem might be deemed “intuitively easy”, although as the number of curves increases, this might not be so easy even for humans; it is also likely to be dependent upon the topological conditions imposed and geometry of the curves used. However, when thinking about a software system we cannot rely on human intuition and have to address the computational issues associated with such apparently simple checks. We have provided a completely new method to solve this problem: we associate a set of marked points with the set of zones and use this to keep track of the zone set of the diagram. We have provided algorithms to update this set of marked points upon the addition of a new curve. Therefore, we can compute the zone set for a given diagram by viewing it as a sequence of curves added one at a time. Also, we can compute the changes in the zone set of one diagram upon the addition of another curve (useful in editing environments). We have also provided evaluation of this new method by presenting the computational improvements over a naive implementation.

The applications of this work are widespread, since it can be utilised in any software systems that utilise Euler diagrams or their extensions. The applications of Euler diagrams and their extensions are steadily growing: they have been used to display the results of database queries [42], for the representation of genetic set relations [30], as logical systems used in automated theorem proving environments [27,38] or for precise software specification (e.g., constraint diagrams [17,28]). Any application area for which the system needs to compute the zone set of a concrete diagram can use these algorithms to improve their efficiency. A major benefit comes when considering an interactive Euler diagram setting. For example, the user may be creating or manipulating these diagrams for some purpose and the system is likely to need to keep track of the information conveyed, perhaps to be able to check if it is a correct deduction in a logical setting, or to be able to use the information to compute with and display the results (e.g., a user might be constructing a diagrammatic database query and want the results displayed, or perhaps even more complex requests such as a set of diagrams to display the changes in data over a certain time period).

In [15] methods for recognising sets of curves whose removal disconnects a diagram were developed, which facilitates the decomposition of problems such as the generation problem into smaller sub-problems for diagrams which have such curves; the integration of these works in the generation area, storing marked points for the diagram pieces to be combined may be beneficial. Another potential application area is that of diagram generation using restricted geometric shapes. For example, a user might not mind if the diagram created does not have exactly the correct set of zones (e.g., one can use

other means such as shading to represent emptiness if necessary), but might really desire to have a diagram drawn with nice geometric shapes (such as circles or ellipses) which has a zone set close to the desired set. We envisage using the ideas in this work to assist in quickly computing diagrams with nice geometric shapes and approximately the correct zone set – for which we can then quickly compute the actual zone set and so indicate to the user what differs from their initial request, for instance. The question of which Venn diagrams can be drawn using convex regular polygons was previously addressed in [2].

In the future, we intend to extend the class of diagrams for which we can apply efficient algorithms utilising the marked points method by relaxing the wellformedness conditions and reducibility. This may be useful when considering expert users of these notations who are not so prone to the errors of comprehension of novices when wellformedness conditions are not enforced, but the complexity of the algorithms is likely to increase in the non-wellformed case.

References

- [1] P.K. Agarwal, M. Sharir, P. Shor, Sharp upper and lower bounds on the length of general Davenport–Schinzel sequences, *Journal of Combinatorial Theory Series A* 52 (2) (November 1989) 228–274.
- [2] J. Carroll, F. Ruskey, M. Weston, Which n -Venn diagrams can be drawn with convex k -gons?, *Discrete & Computational Geometry* 37 (4) (2007) 619–628.
- [3] S.C. Chow, Generating and drawing area-proportional Euler and Venn diagrams, PhD thesis, University of Victoria, 2007.
- [4] S.C. Chow, F. Ruskey, Drawing area-proportional Venn and Euler diagrams, in: *Proceedings of Graph Drawing 2003*, Perugia, Italy, in: LNCS, vol. 2912, Springer-Verlag, September 2003, pp. 466–477.
- [5] R. Clark, Failure mode modular de-composition using spider diagrams, in: *Proceedings of Visual Languages and Logic*, in: CEUR, vol. 274, 2007, pp. 41–54.
- [6] G. Cordasco, R.D. Chiara, A. Fish, Interactive visual classification with Euler diagrams, in: *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing VL/HCC 2009*, IEEE Press, 2009, pp. 185–192.
- [7] R. De Chiara, U. Erra, V. Scarano, VennFS: A Venn diagram file manager, in: *Proceedings of Information Visualisation*, IEEE Computer Society, 2003, pp. 120–126.
- [8] R. De Chiara, 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.
- [9] R. De Chiara, A. Fish, Eulerview: A non-hierarchical visualization component, in: *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society Press, 2007, pp. 145–152.
- [10] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, New York, 1987.
- [11] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, M. Sharir, Arrangements of curves in the plane—topology, combinatorics, and algorithms, *Theoretical Computer Science* 92 (2) (1992) 319–336.
- [12] M. Egenhofer, E. Clementini, P. di Felice, Topological relations between regions with holes, *International Journal of Geographical Information Systems* 8 (2) (1994) 129–144.
- [13] M. Egenhofer, R. Franzosa, On the equivalence of topological relations, *International Journal of Geographical Information Systems* 9 (2) (1995) 133–152.
- [14] M. Egenhofer, Deriving the composition of binary topological relations, *Journal of Visual Languages and Computing* 5 (2) (1994) 133–149.
- [15] A. Fish, J. Flower, Euler diagram decomposition, in: *Proceedings of the 5th International conference on Diagrams 2008*, in: *Lecture Notes in Artificial Intelligence*, vol. 5223, Springer, 2008, pp. 28–44.
- [16] A. Fish, J. Flower, Abstractions of Euler diagrams, in: *Proceedings of the 1st International Workshop on Euler Diagrams*, in: *Electronic Notes in Theoretical Computer Science*, vol. 134, Elsevier, 2005, pp. 77–101.
- [17] A. Fish, J. Flower, J. Howse, The semantics of augmented constraint diagrams, *Journal of Visual Languages and Computing* 16 (2005) 541–573.
- [18] A. Fish, C. John, J. Taylor, A normal form for Euler diagrams with shading, in: *Proceedings of the 5th International Conference on Diagrams 2008*, in: *Lecture Notes in Artificial Intelligence*, vol. 5223, Springer, 2008, pp. 206–221.
- [19] J. Flower, A. Fish, J. Howse, Euler diagram generation, *Journal of Visual Languages and Computing* 19 (6) (2008) 675–694.
- [20] J. Flower, J. Howse, Generating Euler diagrams, in: *Proceedings of 2nd International Conference on the Theory and Application of Diagrams*, Springer, 2002, pp. 61–75.
- [21] J. Flower, J. Howse, J. Taylor, Nesting in Euler diagrams: Syntax, semantics and construction, *Journal of Software and Systems Modelling* 13 (1) (2004) 55–67.
- [22] C.A. Gurr, Effective diagrammatic communication: Syntactic, semantic and pragmatic issues, *Journal of Visual Languages and Computing* 10 (4) (1999) 317–342.
- [23] E. Hammer, S. Shin, Euler's visual logic, *History and Philosophy of Logic* 19 (1) (1998) 1–29.
- [24] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, R.E. Tarjan, Sorting Jordan sequences in linear time using level-linked search trees, *Information and Control* 68 (3) (1986) 170–184.
- [25] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil, Spider diagrams: A diagrammatic reasoning system, *Journal of Visual Languages and Computing* 12 (3) (2001) 299–324.
- [26] J. Howse, S. Schuman, Precise visual modeling: A case-study, *Software and System Modeling* 4 (3) (2005) 310–325.
- [27] J. Howse, G. Stapleton, J. Taylor, Spider diagrams, *LMS Journal of Computation and Mathematics* 8 (2005) 145–194.
- [28] S. Kent, Constraint diagrams: Visualizing invariants in object oriented modelling, in: *Proceedings of OOPSLA97*, ACM Press, 1997, pp. 327–341.
- [29] H. Kestler, A. Muller, T. Gress, M. Buchholz, Generalized Venn diagrams: A new method for visualizing complex genetic set relations, *Bioinformatics* 21 (8) (2005) 1592–1595.
- [30] H. Kestler, A. Muller, H. Liu, D. Kane, B. Zeeberg, J. Weinstein, Euler diagrams for visualizing annotated gene expression data, in: *Proceedings of Euler Diagrams 2005*, 2005.
- [31] H. Kestler, A. Muller, J.M. Kraus, M. Buchholz, T.M. Gress, H. Liu, D. Kane, B. Zeeberg, J. Weinstein, VennMaster: Area-proportional Euler diagrams for functional GO analysis of microarrays, *BMC Bioinformatics* 9 (1) (2008) 67.
- [32] H. Kestler, J. Messner, A. Muller, R. Schuler, On the complexity of intersecting multiple circles for graphical display, in: *Ulmer Informatik-Berichte*, 2008–01, University of Ulm, Germany, 2008.
- [33] D.M. Mount, Geometric intersection, in: J.E. Goodman, J. O'Rourke (Eds.), *The Handbook of Discrete and Computational Geometry*, 2nd ed., Chapman & Hall/CRC, Boca Raton, 2004, pp. 857–876.
- [34] P. Rodgers, L. Zhang, A. Fish, General Euler diagram generation, in: *Proceedings of the 5th International Conference on Diagrams 2008*, in: *Lecture Notes in Artificial Intelligence*, vol. 5223, Springer, 2008, pp. 13–27.
- [35] F. Ruskey, M. Weston, Venn diagrams, *Electronic Journal of Combinatorics* (2005) DS5. Available at www.combinatorics.org/Surveys/ds5/VennEJC.html.

- [36] A. Shimojima, Inferential and expressive capacities of graphical representations: Survey and some generalizations, in: 3rd International Conference on the Theory and Application of Diagrams, Springer, 2004, pp. 18–21.
- [37] G. Stapleton, J. Howse, J. Taylor, A decidable constraint diagram reasoning system, *Journal of Logic and Computation* 15 (6) (2005) 975–1008.
- [38] G. Stapleton, J. Masthoff, J. Flower, A. Fish, J. Southern, Automated theorem proving in Euler diagrams systems, *Journal of Automated Reasoning* 39 (4) (2007) 431–470.
- [39] N. Swoboda, Implementing Euler/Venn reasoning systems, in: M. Anderson, B. Meyer, P. Olivier (Eds.), *Diagrammatic Representation and Reasoning*, Springer-Verlag, 2001.
- [40] 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.
- [41] J. Thièvre, M. Viaud, A. Verroust-Blondet, Using Euler diagrams in traditional library environments, in: *Proceedings of Euler Diagrams 2004*, in: *Electronic Notes in Theoretical Computer Science*, vol. 134, 2005, pp. 189–202.
- [42] 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, 2004, pp. 128–141.
- [43] D. Winterstein, A. Bundy, C. Gurr, Dr. Doodle: A diagrammatic theorem prover, in: *Proceedings of the International Joint Conference on Automated Reasoning*, in: *LNCS*, vol. 3097, Springer, 2004, pp. 331–335.
- [44] D. Winterstein, A. Bundy, C. Gurr, M. Jamnik, Using animation in diagrammatic theorem proving, in: *Proceedings of Second International Conference on Diagrammatic Representation and Inference*, in: *LNCS*, vol. 3097, Springer, 2002, pp. 46–60.