The Online Abstraction Problem for Euler Diagrams

Gennaro Cordasco[1], Rosario De Chiara[2], and Andrew Fish[3]

[1] Dipartimento di Psicologia – Seconda Università di Napoli, ITALY,
gennaro.cordasco@unina2.it
[2] ISISLab, Dipartimento di Informatica – Università di Salerno, ITALY
dechiara@dia.unisa.it
[3] School of Computing, Engineering and Mathematics – University of Brighton, UK
Andrew.Fish@brighton.ac.uk

Abstract. A Euler diagrams are an accessible and effective visualisation of data involving simple set-theoretic relationships. Efficient algorithms to quickly compute the abstract regions of an Euler diagram upon curve addition and removal have been developed, but a strict set of drawing conventions (called wellformedness conditions) were enforced, meaning that some abstract diagrams are not representable as concrete diagrams. We present a variation and extension of the methodology which enables region computations for Euler diagrams under the relaxation of several drawing conventions. We provide complexity analysis and compare with the previous methodology. The algorithms are presented for generic curves, allowing for specialisations such as utilising fixed geometric shapes for curves that often occur in applications.

1 Introduction

Venn [21] and Euler diagrams are a well known representation of sets and their relationships. Venn diagrams have had significant theoretical interest from the likes of Grünbaum and Hamburger in recent times; a detailed survey of Venn diagrams can be found in [16]. Euler diagrams are the modern incarnation of Euler circles [3], first introduced for the purposes of syllogistic reasoning. Whilst Venn diagrams ensure that every region determined by being inside some contours and outside the other contours is nonempty, Euler diagrams generalise Venn diagrams by relaxing this condition. This allows them to specify subset relations and disjointness relations amongst sets without any extra cognitive load since these semantic relationships are well-matched the spatial relationships of containment and disjointness [11, 17].

In a practical setting, Euler diagrams appear frequently in various application domains. For example, they have been used in biological areas for representing complex genetic set relations in [14], in computer-based resource management scenarios in [2], and in the information retrieval/visualisation context to depict the numbers of results of collections of library database query results in [20] and in network visualisation [15]. Euler diagrams, together with diagrammatic inference rules, form a diagrammatic logic, and comparisons of the effect of the choice of inference rules on automated searches for minimal proofs within Euler diagram-based reasoning systems [18] has been investigated. There are many variations of the basic system, and they have also been incorporated into heterogeneous reasoning systems [19]. More complex diagrammatic logics such as Spider [12] or Constraint diagrams [4, 13] build on the underlying Euler diagram logic, adding more syntax in order to increase the expressiveness of the languages.

3rd International Workshop on Euler Diagrams, July 2, 2012, Canterbury, UK.

Motivation. For any computer-based applications there is a natural disparity between the concrete level information that the user perceives and manipulates (the drawn or concrete diagrams) and the abstract information that the system requires or manipulates (the abstract models or abstract diagrams). Many important computations of the system tend to be defined at this abstract level. For instance, if one wished to present the semantics of a user-constructed diagram then the system needs to perform computations such as to identify the regions present in the diagram, to compute the set intersections that they represent, and to combine these into some set-theoretic statement. In a more general sense an efficient way to calculate the abstract diagram is also useful for the comparison of concrete diagrams.

The efficient computation of the abstract model from a given concrete diagram, together with the ability to update the abstract model upon concrete changes such as curve addition, removal, translation and resizing represent an important challenge to be addressed. The relaxation of the drawing constraints is a significant extension because under these relaxed constraints, every abstract diagram has a concrete diagram representing it [5]. Furthermore, for dynamic diagrams (e.g. sequences of diagrams constructed during interactive user construction or the presentation of evolving data sets) it permits the temporary relaxation of the chosen set of drawing conventions imposed for a diagram in the sequence. This enables a natural construction or presentation, assisting in preserving a user's mental map.

Contribution and paper outline. In this paper we provide a solution to the *on-line abstraction problem*: compute the abstraction of a concrete Euler diagram (i.e. a drawn diagram), keep track of the concrete and abstract diagrams, and enable the automatic update of the abstract diagram upon concrete level manipulations. The algorithms presented in [8] solved this problem for the wellformed diagrams of [5], but here we provide a solution for the more general case in which several well-formedness conditions are relaxed, enabling much greater utility and flexibility. The algorithms have also been extended to permit further relaxation of the wellformedness constraints, enabling the processing of 'generalised Euler diagrams' (representing sets as unions of regions with holes), ensuring that any abstract diagram has a concrete realisation. However, this further extension is not presented in this paper due to space constraints.

2 Preliminaries

We provide a definition of Euler diagrams, separating the abstract and concrete models as usual, together with the set of wellformedness conditions considered. Specifically, we incorporate some of the wellformedness conditions of [5] and [10] that are simplicity of contours (no self-intersection) and uniqueness of contour labels, into the main definition of concrete Euler diagram, which enables an omission of labels since the contours can be uniquely identified.

Definition 1. *A* concrete Euler diagram *is a pair* $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ *where* $\mathcal{C}$ *is a set of simple closed curves, called* (concrete) contours*, in the plane and* $\mathcal{Z}$ *is the collection of* (concrete) zones z *determined by being inside a set of contours* $X_z \subseteq \mathcal{C}$ *and outside the rest of the contours. That is,*

$$z = \bigcap_{c \in X_z} int\,(c) \;\cap \bigcap_{c \in \mathcal{C} - X_z} ext\,(c)\,,$$

for each $X_z \subseteq \mathcal{C}$, provided this region is non-empty. Here $int(c)$ and $ext(c)$ denote the interior and the exterior of c, respectively, and the set X_z is called the zone descriptor *for z. A* minimal region *of d is a connected component of $\mathbb{R}^2 - \bigcup_{c \in \mathcal{C}} c$.*

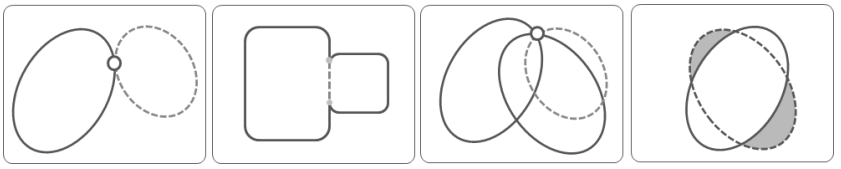

Fig. 1. Nonwellformed Euler diagrams, breaking WF1a, 1b, 2 and 3, resp., from left to right.

We say d is wellformed *if the following wellformedness conditions (WFCs) hold (see Fig. 1):*

WF1 ***Transverse intersections****: Contours that intersect do so transversely.*
This can be subdivided into:
WF1a *No tangential intersections.*
WF1b *No concurrency (distinct contours meet at a discrete set of points).*
WF2 ***No multiple points****: At most two contours can intersect at any given point.*
WF3 ***Connected concrete zones****: Each concrete zone is a* minimal region*.*

An abstract Euler diagram (see Definition 2) is an abstraction (see Definition 3) of a concrete diagram. We overload the term zone, using it for the concrete zones, which are regions of the plane, as well as for abstract zones, which are the sets of containing contours of that region (or the labels of those contours in the generalised case); the context determines which is meant. Let $\mathcal{P}X$ denote the powerset of set X.

Definition 2. *An* abstract Euler diagram *is a pair: $d = \langle C(d), Z(d) \rangle$ where: $C(d)$ is a finite set of labels, called* (abstract) contours*, drawn from some alphabet $\mathcal{L}$. The set of* (abstract) zones *of d is $Z(d) \subseteq \mathcal{P}C(d)$, where $\bigcup_{z \in Z(d)} z = C(d)$.*

Definition 3. *Let d be a concrete Euler diagram and d' an abstract Euler diagram. If there is a bijection between $\mathcal{C}(d)$ and $C(d')$ that induces a bijection between $\mathcal{Z}(d)$ and $Z(d')$, then d is said to be a* realisation *of d', and d' is the* abstraction *of d. An abstract Euler diagram d' is* drawable *if there is a realisation of d' as a concrete Euler diagram.*

By convention, each concrete Euler diagram contains a zone o, called the *outer zone*, which is exterior to all the contours (that is, $X_o = \emptyset$). The left of Fig. 2 shows a concrete Euler diagram containing four contours (A, B, C, D). The zone descriptors for the concrete zones are graphically depicted in the right hand side of the figure; these sets can be viewed as the abstract zone set.

We need terminology relating to the important operations of the addition and removal of the contours of an Euler diagram.

Definition 4. *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a concrete Euler diagram with $A \notin \mathcal{C}$ and $B \in \mathcal{C}$. Let $d + A$ and $d - B$ denote the concrete Euler diagrams obtained by the addition of a new*

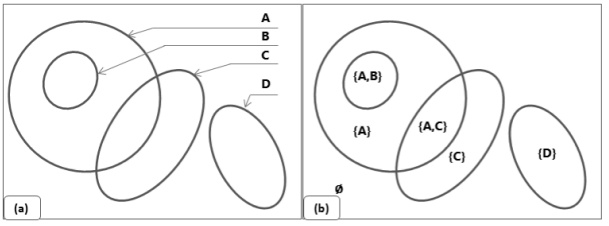

Fig. 2. (a) A wellformed concrete Euler diagram, with contour identifiers (or labels); (b) a depiction of the zone descriptors.

contour A to d and the removal of contour B from d, respectively. A region r of d is a union of minimal regions; it is: (i) a covered region (or is covered *by A) if $r \subset int(A)$ in $d + A$; (ii) split by A (a split region) if $r \cap int(A) \neq \emptyset$ and $r \cap ext(A) \neq \emptyset$ (i.e. r is partially covered by A). Analogously, a zone z of d is a* covered zone *(respectively a split zone) when it is covered (respectively partially covered) by A.*

Fig. 3 shows an example of contour addition. We observe that the zone described by $\{C\}$ is split by the contour A but neither of its two minimal regions is split by A. A point of intersection x between two curves c_1 and c_2 is called a *crossing point*. We denote with $Cross(d)$ the set of all the crossing points generated by contours in d.

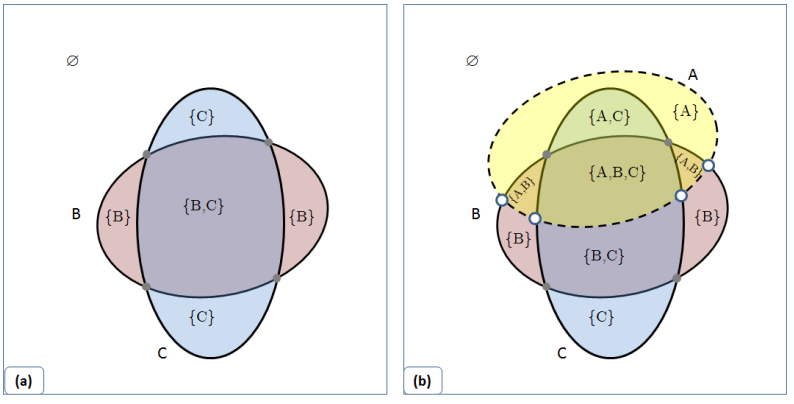

Fig. 3. An example of contour addition: (a) A non wellformed diagram $d = \langle\{B, C\}, \{\emptyset, \{B\}, \{C\}, \{B, C\}\}\rangle$. The crossing points of d are shown with filled-in dots; (b) The crossing points of A with d (i.e. those in $Cross(A)$) are depicted as hollow dots. The set of all hollow and filled-in dots depicts the set of crossing points of $d + A$.

3 Computing the abstraction of Euler Diagrams

The main problem that we address is the following, with variations according to the choice of wellformedness conditions imposed.

Abstraction Update Problem

Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a concrete Euler diagram and $d' = \langle C', Z' \rangle$ the abstraction of d. Let $A \notin \mathcal{C}$ and $B \in \mathcal{C}$. Efficiently compute the abstractions of $d + A$ and $d - B$.

In [6, 8] the *single marked point approach* (SMPA) (described below) was presented, computing the abstraction for wellformed Euler diagrams, following an online approach where diagrams are viewed as a sequence of contour additions and removals. Other operations such as translation or resizing of a contour can be easily simulated by the addition and removal operations, and so such algorithms are applicable in a wider context.

In this paper, we present an evolution of these algorithms, adopting the *multiple marked point approach* (MMPA) (described below), permitting the relaxation of condition WF3 so that Euler diagrams whose zones are disconnected can be processed.

First of all, we recall from [8] that the set of zones split by A, due to the addition of a contour A to (or its removal from) a given wellformed Euler diagram d, can be computed using the following observation.

Observation 1 *Let d be a wellformed Euler diagram and let $\{x_0, x_1, \ldots, x_{m-1}\}$ be all of the crossing points that we meet as we traverse the contour A from an arbitrary point on A. Then:*
(i) For each $i = 0, \ldots, m-1$ each arc $(x_i, x_{i+1 \bmod m})$ splits one zone (note that two arcs can split the same zone but one arc cannot split more than one zone) of d.
(ii) 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 contour (the contour which intersects with A generating the crossing point $x_{i+1 \bmod m}$).

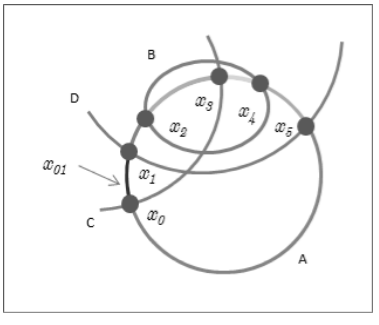

Fig. 4. A schematic diagram which illustrates Observation 1. The addition of A generates six new crossing points shown with small blobs. The point x_{01} is an arbitrary point of the arc (x_0, x_1) used to compute an initial zone descriptor for the zone split by arc (x_0, x_1), whilst subsequent zone descriptors are computed using Observation 1.

For the wellformed diagram case of [8], we adopted the SMPA where each zone z of d has a single point $mp(z) \in \mathbb{R}^2$ associated to it, where $mp(z)$ lay in the boundary of the closure of the zone z (with the possible exception of the marker for the outside zone). These points keep track of the zone sets, and were used to update these sets according to their relationships with contours that are added or removed from a diagram. Then, the zones that are not split by A are covered by A if and only if $mp(z)$ belongs to the interior of A. The SMPA is illustrated in Figure 5: each minimal region is marked by a single marked point (an arrowed dot indicates a marked point, the arrow indicating the

minimal region which is marked); additional marked points, or *pseudo-crossing points*, are used to mark the outside zone and any contour which has no crossing points, either in d or at some stage during its incremental construction (e.g. see the marked point for zone $\{B, D, E\}$ in Figure 5). However, the SMPA is not sufficient to deal with the Euler diagrams with disconnected zones. In this case, there are two ways of splitting a zone: (i) a zone is split when one of its minimal regions is split by A; (ii) a zone is split when some of its constituent minimal regions are covered by A and some are not. To address case (ii) one can consider associating one marked point to each minimal region of the diagram. Figures 5 (c) illustrates a generalization of the case of [8] where each minimal region is associated with one marked point. Then, if a zone z has no minimal regions which are split by A (i.e. case (i) does not hold) we can analyse the relationships of the marked points with A to classify z as split by A, covered by A, or neither. In particular, if all of the marked points of z belong to $int(A)$ then z is a covered zone, whilst if some of the marked points of z belong to $int(A)$ while others do not, then z is a split zone, according to (ii) above.

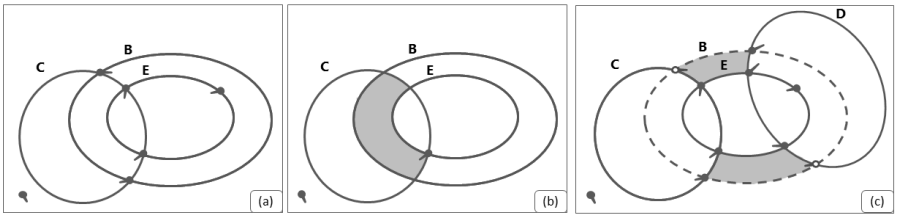

Fig. 5. Single marked point approach (SMPA): in (a) each zone of a wellformed Euler diagram is marked by a single point; (b) shows in grey the zone $\{B, C\}$ and its marked point; in (c) a non wellformed Euler diagram (WF3 relaxed) with a zone $\{B\}$, shown in grey, which consists of two minimal regions, therefore requiring at least two marked points.

However, the management of marked points (taking one for each minimal region) for non-wellformed diagrams (relaxing WF3) raises some tricky problems such as: if a zone z becomes split upon contour addition or deletion, how can one efficiently find a marked point for each of the minimal regions that comprise z? For example, Figure 6 shows two parallel examples which adopt the SMPA (using the algorithm of [8]) in which only the order of contour addiction has been varied. Whilst one of these gives a valid solution, the other does not. In detail, the first two steps (a) and (b) in the figure represent the addition of the first two contours (E and C) to the diagram. Then the two cases are depicted: on the left we add first contour D and then contour B, whilst on the right we first add B and then D. In the first case (on the left) the association between the marked points and minimal regions is correct, with the two minimal regions of zone $\{B\}$ marked by points z_0 and z_2. However, in the second case (on the right) the association is incorrect: there are two marked points associated to the same minimal region (top, shaded) and no marked point associated the other minimal region (bottom, shaded). The problem is that, in general, there is no easy way of discriminating between the case on the left from the case on the right; by just analyzing the relation between a marked point and the contours it is possible to discriminate between zones and not minimal regions.

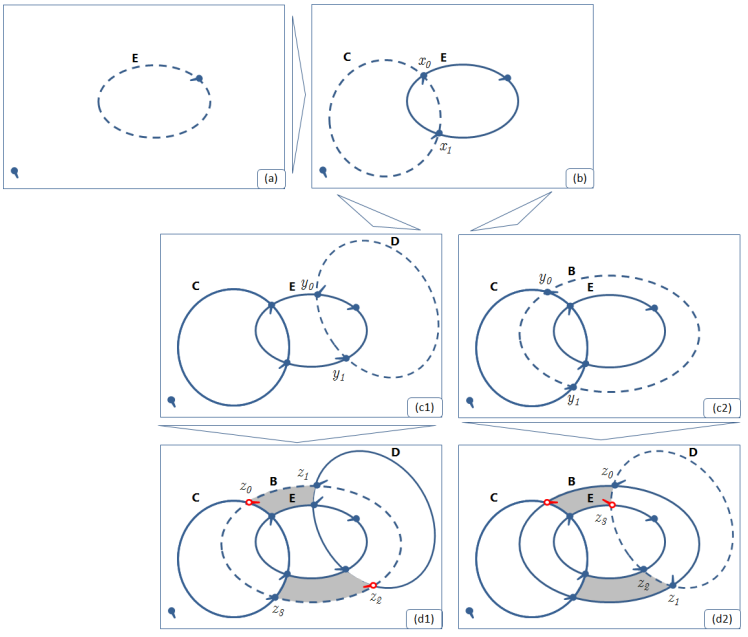

Fig. 6. The influence of the order of contour addition on the marked points/minimal region association, by using the algorithms of [8]. The dotted contour is the one that is going to be added to the current diagram.

We avoid such problems by adopting the MMPA in which each zone is associated with a set of marked points (which comprises the set of crossing points laying on its boundary). This approach requires the tracking of a larger number of marked points but we accept this trade-off against a simpler marked point management (also making implementation easier), since when a zone is split, we just need to correctly partition the set of its marked points.

The MMPA is illustrated in Figure 7: a set of points marks each minimal region r, including all of the crossing points on the boundary of r. Thus, each zone has marked point set including all of the crossing points laying on its boundary (i.e the boundaries of its constituent minimal regions). In the specific case in Figure 7, each marked point marks one, two or four zones.

In the following we define a procedure for contour addition (with contour removal omitted for space reasons), which satisfies:

Theorem 1. *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be an Euler diagram, satisfying WF1 and WF2 (i.e. with WF3 relaxed). Then*

i) *If $A \notin \mathcal{C}$, then the procedure* ***NewContour****(d, A) computes the new collection of zone descriptors for the zones $\mathcal{Z}'$ of $d' = \langle \mathcal{C} \cup A, \mathcal{Z}' \rangle$.*

ii) *If $B \in \mathcal{C}$, then the procedure* ***DeleteContour****(d, B) computes the new collection of zone descriptors for the zones $\mathcal{Z}'$ of $d' = \langle \mathcal{C} - B, \mathcal{Z}' \rangle$.*

Moreover, both procedures:

1. *compute, for each zone $z \in \mathcal{Z}' - \{z_0\}$, where z_0 is the zone in the exterior of all contours in d', the set of marked points of the zone, $MP(z)$, that is comprised of the*

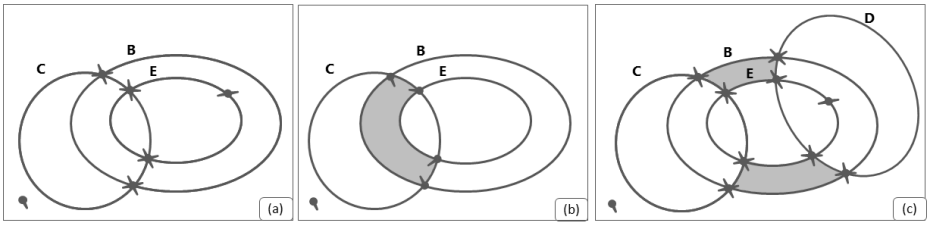

Fig. 7. Multiple marked point approach (MMPA): in (a) each zone is marked by points including all of the crossing points belonging to its boundary; (b) shows in grey the zone $\{B, C\}$ and its marked points; (c) a non wellformed Euler diagram (WF3 relaxed) with a zone $\{B\}$, shown in grey, comprised of two minimal regions, utilising eight marked points.

set of all crossing points (or pseudo-crossing points) of d' belonging to the closure of z. There is a single marked point $mp(z_0)$ in the exterior of all of the curves of d'.
2. *have running time $O(|\mathcal{Z}| + |Cross(d)| \log(|Cross(d)|))$.*

3.1 The algorithms

Initially, we consider the key case of Euler diagrams with WF3 relaxed (but WF2 and WF1 enforced). Subsequently, we will provide the ideas enabling the relaxation of WF2 and WF1. For space reasons we present only the algorithm for contour addition (omitting the algorithm for contour removal). Both algorithms are based on two auxiliary algorithms, **ComputeContourRelationships** (which computes the relationship of a contour with the other contours in a diagram, and updates the marked point sets) and **ComputeSplitRegions** (which computes the zone descriptors of the split zones using Observation 1).

Definition 5. *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a concrete Euler diagram and A a contour which is not in $\mathcal{C}$. Let*

$Over(A)$ denote the collection of all of the contours in $\mathcal{C}$ that properly overlap A; that is $Over(A) = \{c \in \mathcal{C} \mid int(A) \cap int(c) \neq \emptyset\}$;
$Cont(A)$ denote the collection of all of the contours in $\mathcal{C}$ that properly contain A; that is $Cont(A) = \{c \in \mathcal{C} \mid c \notin Over(A)$ and $int(c) \cap int(A) = int(A)\}$.

For example, Figure 3 (a) shows diagram d and Figure 3(b) shows the addition of contour A to d. We have $Over(A) = \{B, C\}$, $Cont(A) = \emptyset$ and $Cross(A)$ contains the four crossing points between A and the contours in $\mathcal{C}$.

The methodology adopted makes use of the following low level computations, and we assume that, given two contours A and B of an Euler diagram with WF3 relaxed, we can quickly find:

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

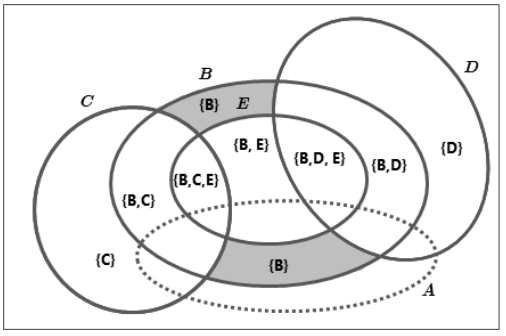

Fig. 8. The addition of contour A splits eight minimal regions determined by the eight arcs that comprise A. However, it splits nine zones, eight of which are the distinct zones containing the eight minimal regions that are split. The ninth zone $\{B\}$ is split, without splitting any of its constituent minimal regions, since one of its minimal regions is covered by A but the other is not.

Placing restrictions on the geometric shapes used for contours (which is common in some applications) can enable particularly fast computations. For example, if each contour 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).

Algorithm **ComputeContourRelationships**: (i) computes the relationship between the contours present in a diagram d (with WF3 relaxed) and a contour A; (ii) updates the set of crossing points of d.

We will refer to Fig. 8 to assist with the explanation of the algorithms. Consider the addition of the dashed contour A to the diagram in Fig. 8 without A. After the execution of Algorithm **ComputeContourRelationships** we have $Cont(A) = \emptyset$, $Over(A) = \{B, C, D, E\}$ whilst $Cross(A)$ is the set of eight crossing points created by the addition of A.

Algorithm **ComputeSplitRegions** uses the sets output by Algorithm **ComputeContourRelationships** and calculates the collection of zone descriptors for the zones that contain a minimal region of $d - A$ split by A. The crossing points of A can be used to decompose A into a set of arcs. This algorithm computes the zone descriptors of all of the zones of $d - A$ that have at least one of their minimal regions split by A.

In detail, the arcs are analysed in the sequence that they are met as one traverses the contour; see Fig. 4 for an example. The region that is split by the first arc (x_0, x_1) is determined the set of contours that properly contain A and then by checking which of the contours that properly overlap with A contains the arc. Each successive region that is split is calculated by computing the difference with the previously computed region; this idea was present in Observation 1.

Contour addition. Algorithm 1 updates the collection of zone descriptors upon the addition of a new contour A to a diagram d. There are two cases to consider. Firstly, if $Over(A) = \emptyset$ then no new crossing points are created by the addition of A, and so A forms a new connected component. Thus, A splits only the zone described by contours

Algorithm 1: NewContour(d, A)

Input: An Euler diagram $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ and a contour A such that $A \notin \mathcal{C}$.
Output: $\mathcal{Z}'$, the collection of zone descriptors of $d' = \langle \mathcal{C} \cup \{A\}, \mathcal{Z}' \rangle$.

$(Cont(A), Over(A), Cross(A)) :=$ **ComputeContourRelationships**(d, A)
if $Over(A) = \emptyset$ **then** // A does not properly overlap any contour present in $\mathcal{C}$
  $s := Cont(A)$ // s is the zone split by A
  $Z_s := \{s\}$ // Z_s is the set of zones having a minimal region split
  $s.points :=$ any point in A // the marked point for s
else
  $Z_s :=$ **ComputeSplitRegions**$(d, A, Cont(A), Over(A), Cross(A))$
$\mathcal{Z}' := \mathcal{Z}$
forall $z \in Z_s$ **do**
  $s := z$ // s is the old zone
  $n := z \cup \{A\}$ // n is the new zone
  $M_s := M_n := M_A := \emptyset$
  forall $x \in s.points$ **do**
    switch do
      case $x \in int(A)$ // the point x marks n
        $M_n := M_n \cup \{x\}$
      case $x \in ext(A)$ // the point x marks s
        $M_s := M_s \cup \{x\}$
      case $x \in A$ // the point x marks both s and n
        $M_A := M_A \cup \{x\}$
  $s.points := M_s \cup M_A$
  $n.points := M_n \cup M_A$
  $\mathcal{Z}' := \mathcal{Z}' \cup \{n\}$ // The new zone n is added to the collection of zones of the diagram
forall $z \in \mathcal{Z} - Z_s$ **do**
  $M_{int} := M_{ext} := \emptyset$ // M_{int} and M_{ext} record the marked points of z that are in the interior and the exterior of A, respectively
  forall $x \in z.points$ **do**
    if $x \in int(A)$ **then**
      $M_{int} := M_{int} \cup \{x\}$
    else
      $M_{ext} := M_{ext} \cup \{x\}$
  if $M_{ext} = \emptyset$ **then** // if z is covered by A then z should be removed
    $\mathcal{Z}' := \mathcal{Z}' - \{z\}$
  else
    $z.points := M_{ext}$
  if $M_{int} \neq \emptyset$ **then** // if z is split or covered by A then a new region should be added
    $n := z \cup \{A\}$
    $n.points := M_{int}$
    $\mathcal{Z}' := \mathcal{Z}' \cup \{n\}$
return $\mathcal{Z}'$

in $Cont(A)$ (in Fig. 2 (a), contour D splits only the outer zone, exterior to all other contours, for example). Secondly, if $Over(A)$ is not empty, then A splits several zones (contour A in Fig. 8 splits several zones, for example). Algorithm 1 computes the split zones in two steps: (i) the split zones which contain a split minimal region are computed by Algorithm **ComputeSplitRegions**; (ii) the split zones which do not have any split minimal regions are computed, together with the set of covered zones, by analysing the relationship between the collection of marked points of the zones and the contour A.

For instance, in Fig. 8, the zones having a minimal region split by A are $\{\emptyset, \{C\}, \{B, C\}, \{B, C, E\}, \{B, E\}, \{B, D, E\}, \{B, D\}, \{D\}\}$ while the zone $\{B\}$ is split by A even though neither of its two minimal regions are split by A since one of them is covered by A and the other is not.

In detail, when $Over(A)$ is empty, A does not properly overlap any of the contours in d and it does not generate any new crossing points. In this case there is exactly one zone of d split, defined by $Cont(A)$, and any point on A can be chosen as the marked point for both of the zones of $d+A$ that are described by $Cont(A)$ and $Cont(A) \cup \{A\}$, referred to as the *old zone* and the *new zone* respectively (lines $3-5$).

Thereafter the algorithm considers the marked points of each zone of d which has a minimal region that is split by A (line 7). The variable s refers to the zone in d that is split as well as the old zone that this becomes upon the addition of A in d', the variable n refers to the new zone that is created in d' from s which is also inside A. For each such marked point x, the algorithm checks if x is a marked point for just the old zone, just the new zone or both (lines $13-20$) in d' (recorded using M_s, M_n and M_A respectively). The new zone n is added to the collection of zones of the diagram (line 23). Subsequently (line 24) the algorithm checks the remaining zones (i.e. those that do not contain any split minimal regions) looking for covered or split zones of d. This is performed by verifying the relationship between the marked points of the zone $z \in \mathcal{Z} - Z_s$ and A (lines $26-30$). In particular, if all of the marked points belong to $int(A)$ (i.e., $M_{ext} = \emptyset$) then the zone z is covered by A and so the old zone z is removed from the diagram (lines $31-32$). Moreover, if at least one marked point belongs to $int(A)$ then the zone z is either split or covered and so a new zone is generated and added to the diagram (lines $35-38$).

3.2 Relaxing the wellformedness condition WF2 and WF1

WF2. We extend the algorithms to also handle crossing points with multiplicity greater than 2 (i.e., more than two contours crossing transversely at a given point). For Euler diagrams with WF3 relaxed, we used Observation 1 in Section 3 to compute the set of zones split by the addition (or removal) of a contour A. Although Observation 1 (i) holds for Euler diagrams with WF2 also relaxed, part (ii) does not hold if the crossing point $x_{i+1 \bmod m}$ has multiplicity greater than 2. Fig. 9 (left) presents a schematic diagram, similar to that in Fig. 4, in which there is a crossing point, x_2, with multiplicity 3. Observation 2 provides the modification of the strategy to deal with diagrams that relax WF2 (as well as WF3).

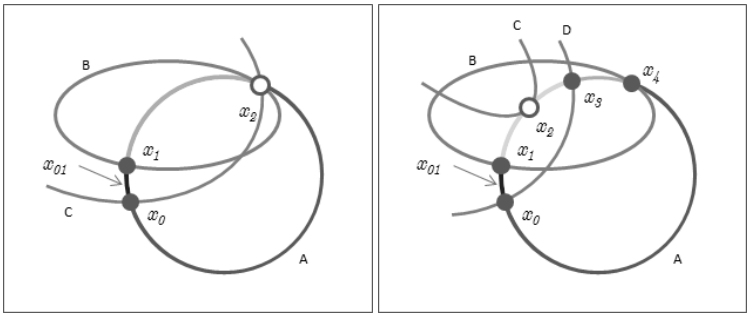

Fig. 9. (left) A schematic diagram illustrating the need for Observation 2 when WF2 is relaxed (c.f. Fig. 4). (right) The addition of A, yielding a diagram with WF1a relaxed. We have four transverse crossing points (shown as filled dots) and one tangential intersection point (shown as a hollow dot).

Observation 2 *Let d be an Euler diagram with WF3 and WF2 relaxed. Let $\{x_0, x_1, \ldots, x_{m-1}\}$ be all of the crossing points that we meet as we traverse the contour A from an arbitrary point on A. If there are exactly $\ell \geq 1$ contours crossing A transversely at a point $x_{i+1 \bmod m}$ then the zone descriptors of the zones that are split by the arcs $(x_i, x_{i+1 \bmod m})$ and $(x_{i+1 \bmod m}, x_{i+2 \bmod m})$ differ by exactly ℓ contours, and these are the contours that intersect with A comprising the crossing point $x_{i+1 \bmod m}$.*

The algorithms in Section 3.1 can now be altered to deal with the relaxation of WF2.

WF1. Firstly, we relax WF1a. Since tangential intersection points do not affect the zones which are split by the corresponding arc (see Fig. 9 (right)), we adapt the algorithm to deal with tangential intersections by simply ignoring them. Hence even the relaxion of WF1a is straightforward.

Secondly, we relax WF1b, allowing concurrency. For this case, we assume that, given two contours A and B of an Euler diagram, we can quickly: (i) check whether they meet in a concurrent arc (i.e. a maximal non-discrete set of points of intersection). (ii) check whether a concurrent arc is tangential (i.e. if a homotopy of the concurrent arc to a point would leave a tangential intersection point; see Figure 10 left) or transversal (i.e. if a homotopy of the concurrent arc to a point would leave a transverse crossing; see Figure 10 right) and (iii) find the split points (the points where two contours that meet in a concurrent arc separate).

The split points will play essentially the same role as the crossing points within the extended algorithms: they will be used as marking points for zones and to compute the set of zones which are split by the addition of a new contour A (noting that the crossing points are also still used, as before). There are two cases to consider: (i) tangential concurrent arcs and (ii) transversal concurrent arcs. For a tangential concurrent arc, both of the split points of that arc do not affect the zones which are split (see Figure 10 left), and so we treat such points in the same manner as tangential intersections; For a transversal concurrent arc, the first split point that we encounter as we traverse the contour A does not affect the zone which is split, whilst the corresponding second split point does affect the zone which is split (see Figure 10 right). Therefore, the first point will be treated as a tangential intersection while the second one will be treated as a transverse crossing.

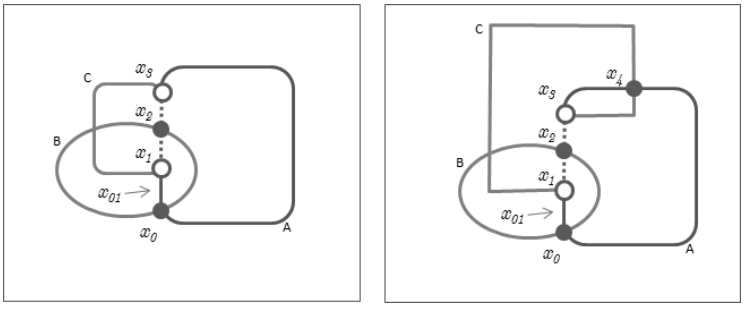

Fig. 10. The addition of A, yielding diagrams with WF1b relaxed. (left) The contour A creates two transverse crossing points, x_0 and x_2, and one tangential concurrent arc between points x_1 and x_3 (shown as hollow dots). (right) The addition of contour A create three transverse crossing points, x_0, x_2 and x_4, and one transverse concurrent arc between points x_1 and x_3 (shown as hollow dots).

3.3 Timing

We provide complexity analysis for the MMPA for Euler diagrams.

The invocation of procedure **ComputeContourRelationships** analyses the relationship between A and each contour in $\mathcal{C}$ and updates the set of crossing points, taking time $O(|\mathcal{C}| + |Cross(d)| \log(|Cross(d)|))$.

Then, if there are intersection points, the procedure **ComputeSplitRegions** computes the set Z_s of zones having a split region and, for each such zone updates the set of marked points. Hence the procedure **ComputeSplitRegions** requires $O(|\mathcal{C}| + |Cross(d)| \log(|Cross(d)|)$ steps.

Finally, for contour addition, lines $9-23$ and $24-38$ respectively compute the collection of marked points for zones having, and not having, split minimal regions, respectively. We can compute this two collections in $O(|\mathcal{Z}| + |Cross(d)|)$ steps. Collectively, algorithm **NewContour** operates within time $O(|\mathcal{Z}| + |Cross(d)| \log(|Cross(d)|))$.

4 Related work and conclusion

The relationship of the Euler diagram abstraction problem with arrangements of Jordan curves in the plane [9] was discussed in [6]. We note that in our approach we do not need to store or manipulate graphs, but we work directly with the diagrams utilising its intersection points and the domain specific data structures, and we obtain a methodology with a straightforward means of implementation.

In [22], an application is presented that interprets an Euler Diagram sketched with a pen or a mouse, and calculates the abstract diagram. The authors claim complexity that is asymptotically similar to ours, but this claim is not substantiated, with the paper not providing details of how the Zone List Refinement Step is performed. The only apparent method that would be effective with the generality that they describe is a pixel based inspection of the drawing (commonly available in programming languages) but which has the drawback of being dependent on the resolution of the image. Our methodology has the added advantage of being more general in that it is not dependent on the image of the contours, but only on their analytical representation.

In [23], a methodology is provided which takes a set of polygons (meaning regions determined by sets of non-overlapping curves) and outputs a set of non-overlapping

polygons, which is essentially the boundary of a zone of the diagram in our terminology; this enables the computation of polygon operations such as union, intersection, difference and clipping. A graph based representation is constructed which consists of a binary tree structure, encapsulating the structure of non-overlapping contours, together with a winged-edge data structure which captures sets of polygons as a graph (indicating the intersection points) and provides a simple means of traversing faces of the graph. Their algorithm "corrects" input containing degeneracies (e.g. zero area contours or coincident edges, meaning concurrency), whereas we wish to develop a method that explicitly considers them. For 'diagrams' that consist of more than one connected component, they compare each output contour with the others to determine if one is inside the area of another or if they co-exist within the same area, and they record these contour relationships in the hierarchical tree structure of the contours. Our approach (utilising marked points) provides removes the need to compute these graph structures and then to operate on them, whilst providing a means to explicitly capture the 'singularities' that may occur within certain contexts or application domains.

In [1], they prove that the Grünbaum encoding uniquely identifies simple Venn diagrams (i.e. they are wellformed) which are monotone and polar symmetric, and develop an algorithm utilising a matrix representation to enumerate the monotone simple symmetric 7-Venn diagrams. The codes considered in the paper rely upon numbering the curves (adopting certain conventions based on the curve segment in other and inner faces to fix the choices) and for a given curve recording the sequence of curve numbers that are given as one traverses the curve. Our methodology applies to a much wider class of diagrams, but investigating the computation of encodings from the data structures utilised in our algorithms is an interesting line of future investigation.

In this paper, we have developed a new methodology, called the multiple marked point approach, which enabled us to develop algorithms to solve the Euler diagram abstraction problem for nonwellformed Euler diagrams relaxing the constraints imposed on the previous state of the art in [8], which utilised the single marked point approach and was limited to the wellformed diagram case. Furthermore, these algorithms extend in a natural manner to enable the processing of generalised Euler diagrams (i.e. unions of regions with holes).

This enables greater flexibility for users of software tools, enabling them to work with nonwellformed diagrams where it is convenient, or necessary, to do so. For example, during user construction of wellformed diagrams if the user is permitted to construct intermediary non-wellformed diagrams then this aids them in adopting a natural construction method, whilst it can be very complex, or even impossible, to do so without allowing passage through non wellformed diagrams. Furthermore, the extension to generalised Euler diagrams ensures that every abstract diagram has a concrete realisation (which is not the case for wellformed diagrams, as shown in [5]).

The algorithms presented in this paper are available in a Java library, although to simplify implementation (and to improve the efficiency) it restricts the geometric shape of contours to be arbitrarily rotated ellipses [7]. This enables the specification of each contour in parametric form, enabling easy implementation of operations such as choosing a point on an arc, or checking contour relationships of intersection or containment.

The applications of this work are widespread since the algorithms can be utilised in any software system that utilise Euler diagrams or their extensions.

References

1. T. Cao, K. Mamakani and F. Ruskey. Symmetric Monotone Venn Diagrams with Seven Curves. In *Fifth Intern. Conference on Fun with Algorithms*, LNCS 6099, 331-342, 2010.
2. R. De Chiara, U. Erra, and V. Scarano. VennFS: A Venn diagram file manager. In *Proceedings of Information Visualisation*, pages 120–126. IEEE Computer Society, 2003.
3. L. Euler. Lettres a une princesse dallemagne sur divers sujets de physique et de philosophie. *Letters*, 2:102–108, 1775. Berne, Socit Typographique.
4. A. Fish, J. Flower, and J. Howse. The Semantics of Augmented Constraint Diagrams. *Journal of Visual Languages and Computing*, 16:541–573, 2005.
5. J. Flower, A. Fish, and J. Howse. Euler diagram generation. *Journal of Visual Languages and Computing*, 19:675–694, 2008.
6. G. Cordasco, R. De Chiara, and A. Fish. Interactive Visual Classification with Euler Diagrams. In *Proceedings of VL/HCC*, pages 185–192. IEEE Press, 2009.
7. G. Cordasco, R. De Chiara, and A. Fish. EulerDiagramNWF: source code. http://isis.dia.unisa.it/projects/EulerNWF, 2010.
8. G. Cordasco, R. De Chiara, and A. Fish. Efficient on–line algorithms for Euler diagram region computation. *Comput. Geometry: Theory and Application (CGTA)*,44:52–68, 2011.
9. H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel and M. Sharir. Arrangements of curves in the plane—topology, combinatorics, and algorithms. In *Theoretical Computer Science*, Vol. 92 N. 2, pages 319–336. Elsevier Science Publishers Ltd. 1992.
10. A. Fish. Euler Diagram Transformations. *Graph Transformations & Visual Modelling Techniques, ECEASST*, 18:1–17, 2009.
11. C. A. Gurr. Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. *Journal of Visual Languages and Computing*, 10:317–342, 1999.
12. J. Howse, F. Molina, J. Taylor, S. Kent, and J. Gil. Spider Diagrams: A Diagrammatic Reasoning System. *Journal of Visual Languages and Computing*, 12(3):299–324, 2001.
13. S. Kent. Constraint Diagrams: Visualizing Invariants in Object Oriented Models. In *Proceedings of OOPSLA97*, pages 327–341. ACM Press, 1997.
14. H. Kestler, A. Muller, T. Gress, and M. Buchholz. Generalized Venn diagrams: A new method for visualizing complex genetic set relations. *J. of Bioinformatics*, 21(8), 2005.
15. N. Henry Riche, and T. Dwyer. Untangling Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010.
16. F. Ruskey. A survey of Venn diagrams. *Electronic Journal of Combinatorics*. www.combinatorics.org/Surveys/ds5/VennEJC.html, 1997
17. A. Shimojima. Inferential and Expressive Capacities of Graphical Representations: Survey and Some Generalizations. In *Proceedings of 3rd International Conference on the Theory and Application of Diagrams*, Vol. 2980 of *LNAI*, pages 18–21. Springer-Verlag, 2004.
18. G. Stapleton, J. Masthoff, J. Flower, A. Fish, and J. Southern. Automated Theorem Proving in Euler Diagrams Systems. *Journal of Automated Reasoning*, 39:431–470, 2007.
19. N. Swoboda, and 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):136–149, 2004.
20. J. Thièvre, M. Viaud, and A. Verroust-Blondet. Using Euler Diagrams in Traditional Library Environments. In *Euler Diagrams 2004*, Vol. 134 of *ENTCS*, pages 189–202, 2005.
21. J. Venn. On the Diagrammatic and Mechanical Representation of Propositions and Reasonings. *Phil.Mag*, 1880.
22. M. Wang, B. Plimmer, P. Schmieder, G. Stapleton, P. Rodgers, and A. Delaney SketchSet: Creating Euler diagrams using pen or mouse. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing VL/HCC*, pages 75–82. IEEE Press, 2011.
23. K. Weiler. Polygon comparison using a graph representation. *Computer Graphics (SIGGRAPH '80 Proceedings)*, 14(3):10–18, July 1980.