Journal of Visual Languages and Computing xx (xxxx) xxxx–xxxx

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

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

Online region computations for Euler diagrams with relaxed drawing conventions

Gennaro Cordasco[a], Rosario De Chiara[b], Andrew Fish[c,*]

[a] Dipartimento di Psicologia – Seconda Università di Napoli, Italy
[b] Poste Italiane – Software Factory Napoli, Italy
[c] School of Computing, Engineering and Mathematics – University of Brighton, UK

ARTICLE INFO

Keywords:
Euler diagrams
Region computation
On-line algorithms
Interactive Diagram Construction

ABSTRACT

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 previously been developed (the single marked point approach, SMPA), but a strict set of drawing conventions (called well-formedness conditions) were enforced, meaning that some abstract diagrams are not representable as concrete diagrams. We present a new methodology (the multiple marked point approach, MMPA) enabling online region computation for Euler diagrams under the relaxation of the drawing convention that zones must be connected regions. Furthermore, we indicate how to extend the methods to deal with the relaxation of any of the drawing conventions, with the use of concurrent line segments case being of particular importance. We provide complexity analysis and compare the MMPA with the SMPA. We show that these methods are theoretically no worse than other comparators, whilst our methods apply to any case, and are likely to be faster in practise due to their online nature. The machinery developed for the concurrency case could be of use in Euler diagram drawing techniques (in the context of the Euler Graph), and in computer graphics (e.g. the development of an advanced variation of a winged edge data structure that deals with concurrency). The algorithms are presented for generic curves; specialisations such as utilising fixed geometric shapes for curves may occur in applications which can enhance capabilities for fast computations of the algorithms' input structures. We provide an implementation of these algorithms, utilising ellipses, and provide time-based experimental data for benchmarking purposes.

1. Introduction

Venn [38] 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 [30]. Euler diagrams are the modern incarnation of Euler circles [18], 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 present (i.e. is a nonempty region of the plane), 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 to the spatial relationships of containment and disjointness, and they give rise to free, or cheap, rides [22,31].

In a practical setting, Euler diagrams appear frequently in various application domains. For example, they have been used in biological areag for representing complex genetic set relations in [26], in computer-based resource management scenarios in [14], and in the information retrieval/visualisation context to depict the numbers of results of collections of library database query results in [37] and in network visualisation [28]. 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 [33] has been investigated. There are many variations of the basic system, and they have also been incorporated into heterogeneous reasoning systems [36]. More complex diagrammatic logics such as Spider [24] or Constraint diagrams [19,25] build on the underlying Euler diagram logic, adding more syntax in order to increase the expressiveness of the languages.

* Corresponding author.
E-mail addresses: gennaro.cordasco@unina2.it (G. Cordasco), rosario@dechiara.eu (R. De Chiara), Andrew.Fish@brighton.ac.uk (A. Fish).

http://dx.doi.org/10.1016/j.jvlc.2016.10.006
Received 11 October 2015; Received in revised form 28 May 2016; Accepted 19 October 2016
Available online xxxx
1045-926X/

1.1. Motivation

For any computer-based application 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). This is because much of the geometric information (e.g. the type of shape of the contours, or the actual positions of the points of the contours) encoded is not being utilised in the abstract model; the abstract model could be viewed as forgetting the information that is not relevant to the semantics. 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 a set-theoretic statement. In a more general sense, an efficient method to calculate the abstract diagrams of concrete diagrams is additionally useful in enabling a fast comparison of semantically important features of concrete diagrams.

In an interactive setting, where users may manipulate the concrete diagrams, the system needs to be able to update the abstract model in accordance with user interaction (e.g. adding a new contour). Since it is also desirable to store the diagram created by the user, an interactive Euler diagram based system should ideally be able to: (i) compute the abstract information quickly; (ii) update this information upon changes to the concrete information; (iii) store the concrete information. Another example arises in the diagrammatic logic setting, where computations (diagrammatic logical inference rules) occur at the abstract level, although a user again sees a graphical interface. If a user sees a diagram and wishes to apply a diagrammatic inference rule, then the computation can involve computing the abstract model, performing the appropriate inference rule and then redrawing or updating the diagram appropriately, where it is possible to do so. However under some sets of well-formedness conditions not all abstract Euler diagrams are drawable, as shown in [21]. Relaxing well-formedness conditions may enlarge the class of abstract diagrams that are drawable. Even if one wishes to primarily focus on well-formed diagrams, transformations between well-formed diagrams may be realisable as sequences of intermediary diagrams that are not necessarily well-formed diagrams. Fig. 1 shows an example involving the relaxation of drawing conventions and the use of highlighting breaks in well-formedness, whilst Fig. 2 show examples requiring the relaxation of drawing conventions. The presentation of such examples motivates the need to consider the relaxation of the drawing conventions in practise, whether tools are being produced for automating drawings or for interpreting diagrams. An example of a tool for which a solution to the online abstraction problem was essential is FunEuler [12] (see Fig. 3).

We envisage the use of the techniques presented here in a more general setting, within which the manipulation of a concrete diagram may occur. A motivating example is provided in [5], where an experimental prototype has been implemented to evaluate the feasibility of Σ Query Language (ΣQL) [4,41] techniques. The ΣQL is an extension of SQL, attempting to address the problem of representing spatial/temporal queries in a natural manner. In [6,7], a query system for ΣQL is described in detail, describing query processing, refinement and optimization. The ΣQL was designed with the intent of supporting the development of information retrieval systems, where a tool for building visual queries is of great significance. In ΣQL there are three families of operators: a spatial operator, a direction operator and a temporal operator. The spatial predicates help to specify the relation between two objects and, in the original paper [5] they are referred to as *disjoin*, *meet*, *overlap*, *coveredBy* and *inside* (see Fig. 4; the term disjoint in more commonly used than disjoin for this relation). Such predicates can be easily verified by using the algorithms described in this paper (given the stated assumptions on the computation of the intersection points): each of the polygons in the database on which the user intends to perform queries is represented by a curve in a concrete Euler Diagram; each of the predicates can be verified by checking the existence of certain zones and/or the cardinality of intersection points. For example, the *disjoin* (the first case from the left in Fig. 4) predicate between generic polygons A and B can be verified by adding the polygons in an Euler Diagram and checking whether or not the zone $\{A, B\}$ is present; the *meet* (the second case in Fig. 4) predicate can be checked by verifying that a single intersection point between A and B, that is a tangential intersection, exists in the concrete diagram.

Therefore, 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 represents an important challenge to be addressed. The ability to solve this problem, whilst permitting the relaxation of the drawing constraints (i.e. the well-formedness conditions), is a significant extension, since under these relaxed constraints, every abstract diagram has a concrete diagram representing it (and concurrent line segments are generally considered as troublesome to deal with). For dynamic diagrams (e.g. sequences of diagrams constructed during interactive user constructions or as the presentation of evolving data sets) it permits the temporary relaxation of the chosen set of drawing conventions imposed on diagrams within the sequence. This enables a natural construction or presentation by assisting in the preservation of a user's mental map.

1.2. Contribution and paper outline

In this paper, we provide a new 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. That is, given a concrete diagram d, consisting of a

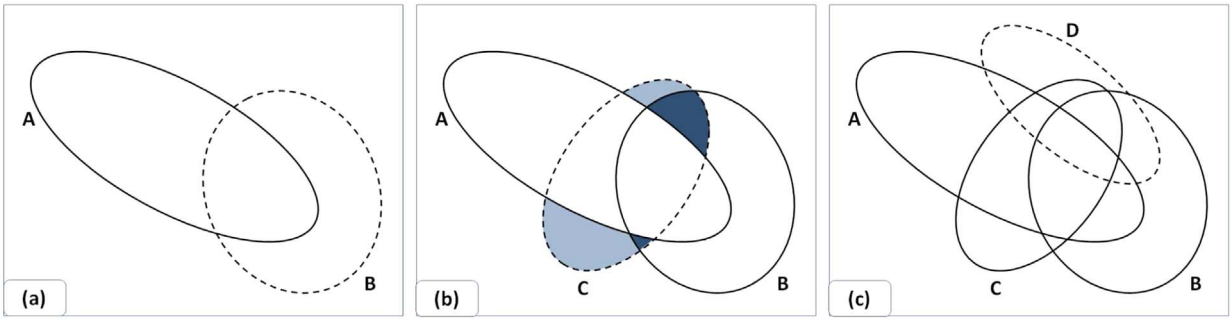

Fig. 1. The construction of a well-formed Euler diagram as a sequence of contour additions that passes through diagrams that break drawing conventions. The highlighted regions (in the middle image) indicate zones which are not connected (i.e. which are comprised of multiple minimal regions), in order to draw the users attention to this fact to try to reduce the potential for human reasoning errors.

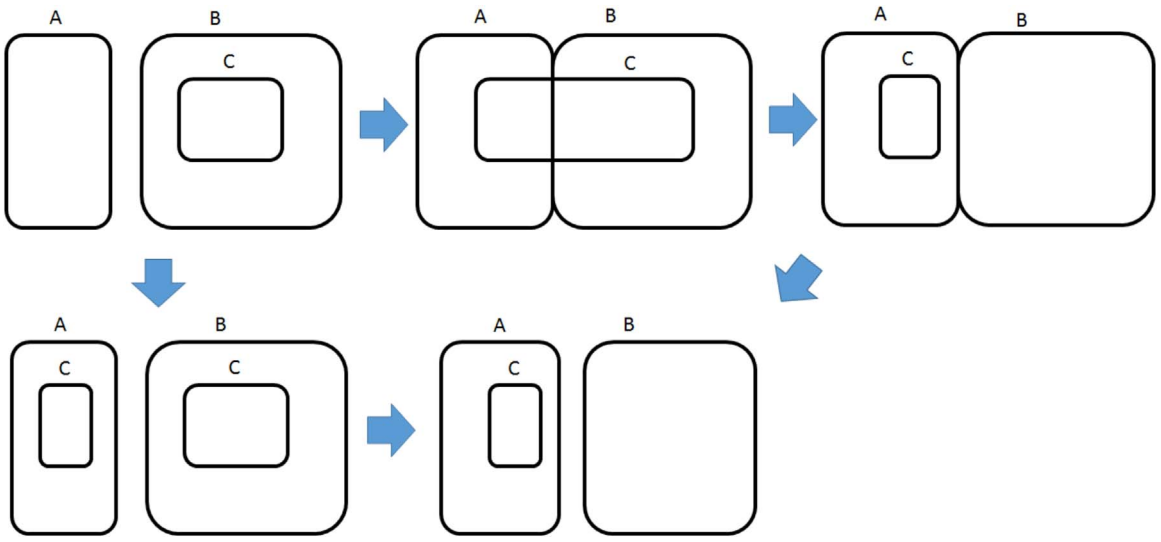

Fig. 2. Consider the presentation of a dynamic set based data input source, depicting changes in the underlying data over time. Suppose that we have two populations *A*,*B*, and initially *C* is a sub-population of *B*, but over time all of the elements of *C* migrate from inside *B* until they are all inside *A*. If any intermediate state is shown in which some of the members of *C* are in *A* and some are in *B*, then the relaxation of drawing conventions is required. The top path shows a sequence of Euler diagrams that break WF1 (using concurrent lines), whilst the bottom path uses generalised Euler diagrams (two curves with the same label).

collection of contours $C(d)$, (i) compute the collection of abstract zones $\mathcal{Z}(d)$ for $C(d)$ and (ii) efficiently update this collection upon the addition/removal of contours to/from $C(d)$. Since operations, such as translation or resizing of a contour, can be simulated by addition and removal of contour operations (e.g. the translation of a contour can be simulated via the removal of the contour followed by the addition of a new contour at the desired location), the algorithms are applicable in a wider context.

The algorithms presented in [13] solved the online abstraction problem for the well-formed diagrams of [21], adopting a single marked point approach (SMPA) but here we provide an alternative solution, adopting the multiple marked point approach (MMPA). This addresses the general case in which the well-formedness conditions are relaxed, enabling much greater utility and flexibility as well as ensuring that any abstract diagram has a concrete realisation. We present the

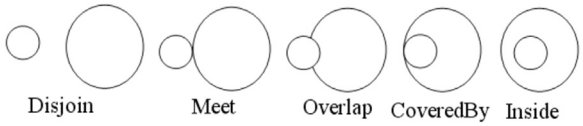

Fig. 4. The spatial predicates from the ΣQL [47].

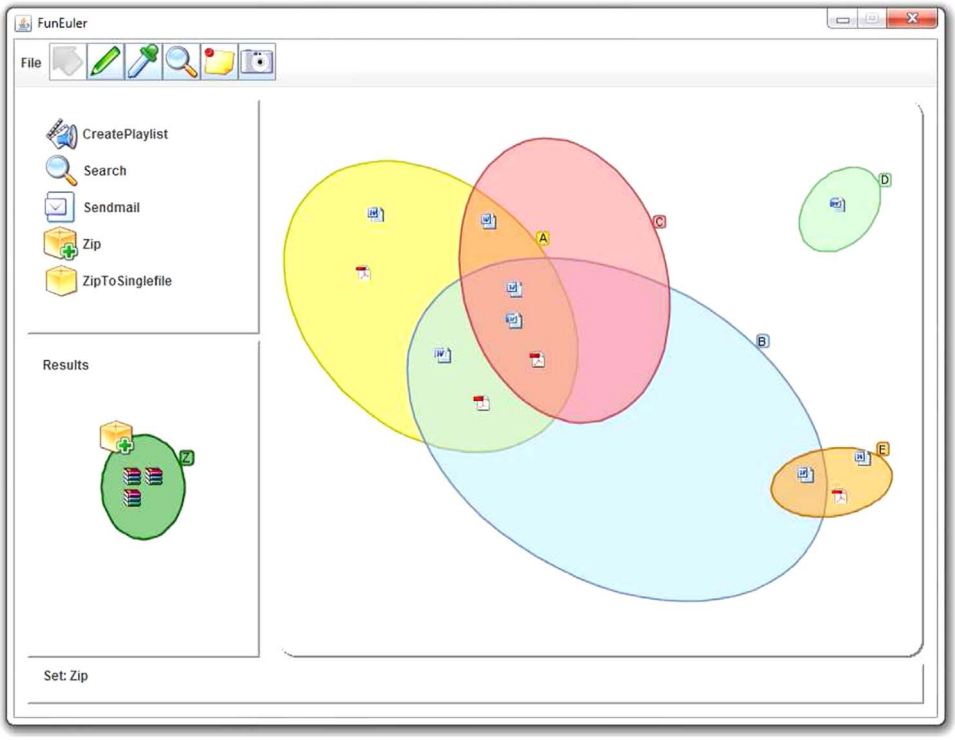

Fig. 3. The FunEuler interface. Contours can be drawn by the user (in the right hand pane) to specify a structured set of queries (one for each zone presented). The icons in top left hand pane represent functions that can be “drag and dropped” onto a region, thereby applying the function to the set of items retrieved by the associated query. Efficient identification of the diagram's regions are essential in this context.

algorithm using a modular approach. First of all we introduce MMPA, enabling the relaxation of WF3, in Section 3, observing why this is a problematic case for the SMPA. Then, additionally, we present two independent extensions that can be used to additionally relax either WF1, WF2 or both of them (and they could be applied to the SMPA as well as the MMPA). This paper is a significantly expanded version of [11], where an outline of the extension for one particular WF condition was presented, without formal detail.

Complexity analysis enables a comparison with the existing methodology (SMPA) for the well-formed diagram case. The algorithms presented in this paper work for generic curves, whilst allowing for specialisations such as imposing constraints on the shape of the curves. Another advantage of our methodology is that the implementation of the algorithms is straightforward. We provide time-based data from a specialised implementation utilising ellipses; whilst the use of ellipses is not required for our approach, the geometric constraints imposed means that the identification of the intersection points is achievable efficiently.

In detail, in Section 2, we provide preliminaries, giving background notions required (concrete and abstract diagrams, the well-formedness conditions and covered or split regions). Section 2.1 present concepts required for the consideration of concurrency. In Section 3, we provide the (MMPA) methodology for solving the abstraction problem for the case of diagrams that have disconnected zones (see WF3 in Section 2), and indicate how it differs from the SMPA approach for the well-formed diagram case of [13]. We present detailed algorithms for contour addition, whilst the technical detail of contour removal is deferred to Section 3.2. Sections 3.3 and 3.4 deal with the cases of relaxing WF2 and WF1 within the presented methodology. The ability to deal with WF3 (disconnected zones) is important in order to be able to highlight disjoint regions that represent the same set intersection, for instance. The ability to deal with WF1b (concurrent line segments) is particularly interesting since other related areas (c.f. Euler graph, winged edge structures) struggle to deal with this case. In Section 4, we relax two conditions that we incorporated into the main definition of Euler diagrams to simplify notation (allowing multiple curves with the same label or non-simple curves), adopting the name generalised Euler diagrams is one permits the relaxation of these conditions. Then, in Section 5, we provide a result relating the number of split points and crossing points to the number of minimal regions present in any diagram, yielding a simple check if a diagram contains disconnected zones from this information. To provide an indication of the efficiency of the algorithms in practise, a prototype tool has been developed which realises the algorithms presented. The interface permits the use of ellipses for contours, within the context of an application for resource management, and we present details and benchmarking in Section 6. Discussions of related work, conclusions and further work are provided in Sections 7 and 8.

Fig. 5 gives an overview of the conditions imposed in diagrams, the problem addressed and the structure of the paper.

2. Preliminaries

We provide a definition of Euler diagrams, separating the abstract and concrete models as usual, together with the set of well-formedness conditions considered. For readability purposes, we incorporate some of the well-formedness conditions of [21] and [20] into the basic definition of an Euler diagram in this paper. This enables us to reduce the notation used for the main body of work. To demonstrate that the methods extend to the cases where these conditions are relaxed, in Section 4 we deal with these 'generalised Euler diagrams'. Specifically, we incorporate the simplicity of contours (no self-intersection) and uniqueness of contour labels into the main definition of concrete Euler diagrams. In this case, there is a natural correspondence between concrete contour identifiers and labels, and adopting this view separates the concerns over labels used for semantics and those used for contour identification; this makes the conceptual transition to the generalised case with non-unique labels, in Section 4, simpler.

Definition 1. A *concrete Euler diagram* is a pair $d = \langle C, \mathcal{Z} \rangle$ where:

1. C is a set of labelled simple closed curves, called (concrete) contours, in the plane, with labels drawn from some given alphabet $\mathcal{L}$, and.
2. $\mathcal{Z}$ is the collection of (concrete) zones z determined by being inside a set of contours $X_z \subseteq C$ and outside the rest of the contours. That is,

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

for each $X_z \subseteq C$, provided this region is non-empty.

Here $int(c)$ and $ext(c)$ denote the interior and the exterior of c, respectively (these are the sets of points in the two regions of $\mathbb{R}^2 - \{c\}$), and the set X_z is called the *zone descriptor* for z. A *minimal region* of d is a connected component of $\mathbb{R}^2 - \bigcup_{c \in C} c$.

We say d is *well-formed* if the following well-formedness conditions (WFCs) hold: .**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*.

We include the set of zones in Definition 1, despite being derivable from the set of contours, since their explicit expression is helpful in the presentation of the algorithms, and to make the connection between the concrete and abstract definitions more transparent. 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); 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 label-preserving bijection between $C(d)$ 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 d.

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. 6 shows a concrete Euler diagram containing four contours with contour identifiers (equivalently, labels) depicted utilising some arrows that are not part of the diagram. 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. Fig. 7 shows examples of violation of the well-formedness conditions.

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

Definition 4. Let $d = \langle C, \mathcal{Z} \rangle$ be a concrete Euler diagram with $A \notin C$ and $B \in C$. Let $d + A$ and $d - B$ denote the concrete Euler diagrams obtained by the addition of a new 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. 8 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

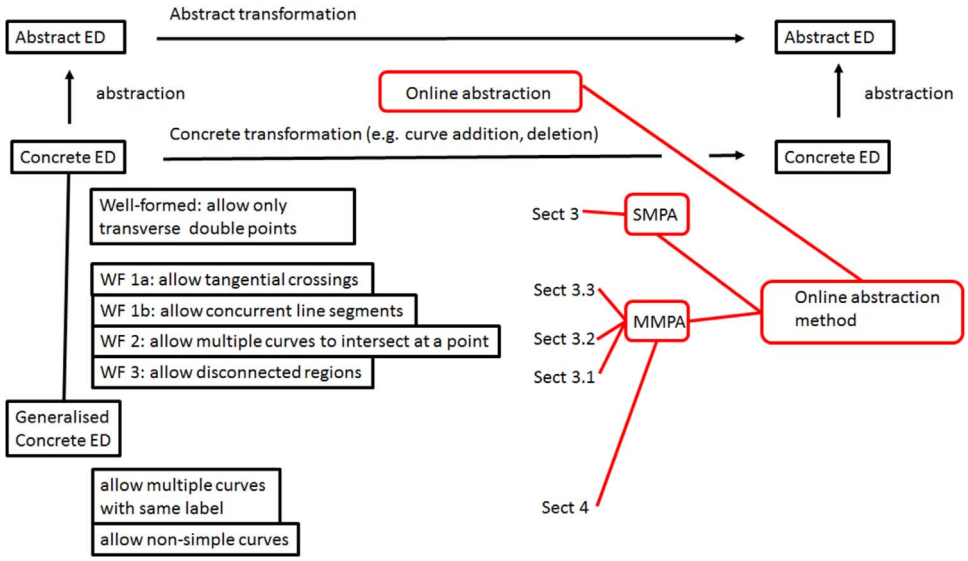

Fig. 5. An overview figure, indicating the online abstraction problem which generalises the static abstraction problem of computing the abstract model of a concrete diagram; it requires the tracking of both abstract and concrete diagrams, whilst enabling abstract model update in accordance with concrete model changes. We recall the single marked point approach (SMPA) in Section 3 that works for well-formed Euler diagrams, but this method does not work if the well-formedness conditions are relaxed. So we introduce a new method, the multiple marked point approach (MMPA) that does works for the relaxation of WF3 in Section 3.1. We also demonstrate how to adapt the algorithms to relax WF2 and WF1, with methods that can be applied independently or together. Finally, we demonstrate that the approach works when generalising the notion of Euler diagrams.

minimal regions is split by A. Analogously, for contour removal, we say that a zone z is: (i) covered by B if $r \subset int(B)$ in d; (ii) split by a contour B in d if the zone z of $d - B$ is split by the addition of contour B.

2.1. Refining intersection point types for the concurrency case

We extend the concepts of intersection or crossing points, developing new concepts of split points to deal with the relaxation of the well-formedness conditions. Fig. 11 shows examples demonstrating the idea, and we formalise these concepts in the following. First of all, we need a basic topological notion of 'local', intuitively considering a small region around a point of interest, by choosing a small value for ϵ in Definition 5.

Definition 5. Let x be a point on an Euler Diagram d (i.e. a point on any curve in $C(d)$). An ϵ- neighbourhood of x, for $\epsilon > 0$, is $B_\epsilon(x) = \{a \in \mathbb{R}^2 : |x - a| < \epsilon\}$, a ball of radius ϵ around x.

We need to distinguish between points of intersection between curves (formerly called intersection points) that cross, either tangentially or transversely, which will be called crossing points (see Definition 6), and points of the curves that are the ends of part of shared segments of the curves (i.e. concurrent arcs), named split points. The crossing multiplicity of any point x on an Euler Diagram will be the maximal number of non-concurrent arcs that pass through x (see Definition 9). Fig. 8 shows *Cross*(A)and the set of eight crossing points of $d + A$, each of which has multiplicity 2.

Definition 6. A point of intersection x between two curves c_1 and c_2 is called a crossing point if there is an $\epsilon > 0$ such that there are no other points of intersection between c_1 and c_2 within an ϵ-neighbourhood of x. Let d be a diagram and $A \notin C(d)$. If x is a crossing point between contour A and any contour in $C(d)$, then it is called a *crossing point* of A with d. The set of all of crossing points of A with d is denoted by *Cross*(A). The set of *crossing points* of d, *Cross*(d), is the union of all of the crossing points between its contours.

In Definitions 9 and 10 we identify points on the set of curves in a diagram at which concurrent arcs separate, and define the *splitting number* of any such a point to account for the variations in the vertex degree of the associated graph of the diagram at points for which the concurrent arcs separate, when applying Euler's formula in Section 5. We first provide terminology to distinguish different types of local concurrency at a point x: total local concurrency where two contours coincide along a segment (an interval) through x, or local partial concurrency where they coincide along one segment approaching x but they separate at x. If they locally meet only at a discrete set of points they are called locally non-concurrent. We also need the concept of a contour c_j being locally concurrent within a set of contours C_I, intuitively meaning that c_j is always locally concurrent with some $c_i \in C_I$, but the c_i can vary..

Definition 7. Let x be a point of an Euler Diagram. Two contours c_{i_1}

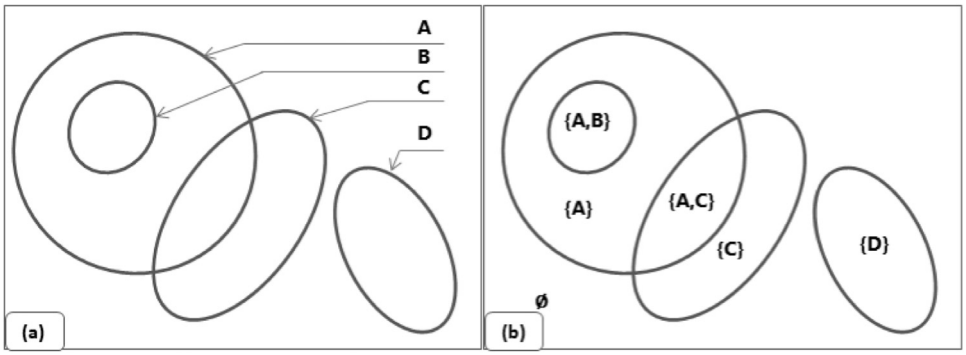

Fig. 6. (a) A well-formed concrete Euler diagram, with contour identifiers (or labels); (b) a depiction of the zone descriptors. The concrete Euler diagram is a realisation of the abstract Euler diagram $\langle \{A, B, C, D\}, \{\emptyset, \{A\}, \{C\}, \{D\}, \{A, B\}, \{A, C\}\}\rangle$.

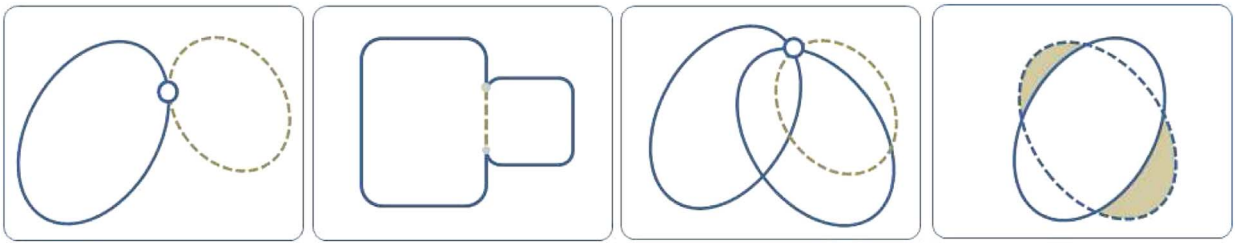

Fig. 7. Non-well-formed Euler diagrams, breaking WF1a, 1b, 2 and 3, respectively, from left to right.

and c_{i_2} are:

1. *locally non-concurrent* at x if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap c_{i_1} \cap c_{i_2}$ is a discrete set, and *locally concurrent* otherwise.
2. *locally completely concurrent* at x if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap c_{i_1} = B_\epsilon(x) \cap c_{i_2}$.
3. *locally partially concurrent* at x if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap c_{i_1} \cap c_{i_2}$ is a radius of $B_\epsilon(x)$ (i.e. it is comprised of a line from the centre x to the boundary of the ball).

A contour c_j is locally concurrent within a set C_I of contours if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap c_j \subseteq B_\epsilon(x) \cap \bigcup_{i \in I} c_i$. Similarly, an intersection of contours $c_{j_1} \cap c_{j_2}$ is locally concurrent within a set C_I of contours if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap (c_{j_1} \cap c_{j_2}) \subseteq B_\epsilon(x) \cap \bigcup_{i \in I} c_i$.

Definition 8. Suppose that two contours c_{i_1} and c_{i_2} of an Euler Diagram are not equal but they are locally completely concurrent at x. Let I be the segment (or arc) of $c_{i_1} \cap c_{i_2}$ that contains x, and let y, z denote the endpoints of this segment. Then I is said to be *tangential* or *transversal* if a homotopy of I to a point would leave a tangential or transverse intersection point, respectively.

Note that c_{i_1} and c_{i_2} in Definition 8 are *locally partially concurrent* at y and z by definition, and y, z will be called split points. See Fig. 9 for an example of tangential or transversal segments. These, together with the orientation of traversal, will be used to determine if the split points should behave as per tangential crossing points or tangential crossing points during the algorithms, later on.

Definition 9. Let x be a point on an Euler Diagram d. A *maximal intersecting contour set for* x is a maximal set of contours $C_I \subseteq C(d)$ such that x is a point on c_i for each contour $c_i \in C_I$, and the contours in C_I are pairwise locally non-concurrent.

The point x has *crossing multiplicity* $i = |C_I|$ (or just *multiplicity* for short), written $m(x) = i$. We take $m(x) = 1$ if x is not a crossing point.

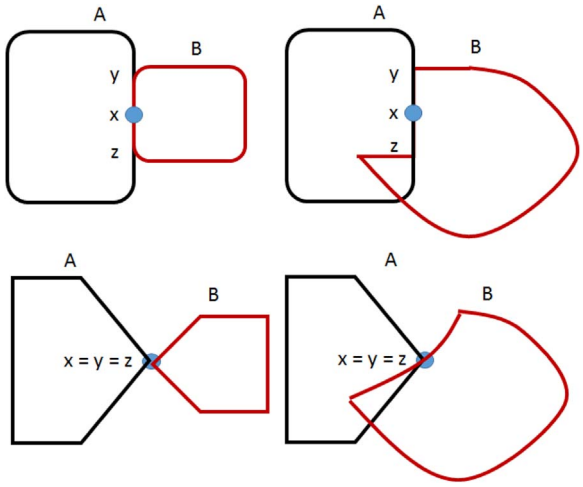

Fig. 9. The top left shows a pair of contours A, B, which are locally totally concurrent at x, and locally partially concurrent at the split points y,z. The segment I between y and z is tangential, since the effect of contracting I to a point leaves a tangential intersection point, as shown at the bottom left. Similarly we see that the corresponding segment in the top right diagram is transverse by considering the contracting of the segment leading to the diagram at the bottom right.

If no pair of curves in $C(d)$ intersect then C_I is any single contour than contains x. In Definition 9, maximal is taken to mean that there is no larger such set, rather than that no more contours can be added to a given set with the required properties. These are different, as demonstrated by Fig. 10 which shows an Euler Diagram at the top of the figure along with two choices of contour sets intersecting at x, of size three of the left, and two on the right. Given that there are four contours and one cannot include B together with either C or D, we see that three is the largest such number and $\{A, C, D\}$ is a maximal intersecting contour set with $m(x) = 3$. The intuition of multiplicity is

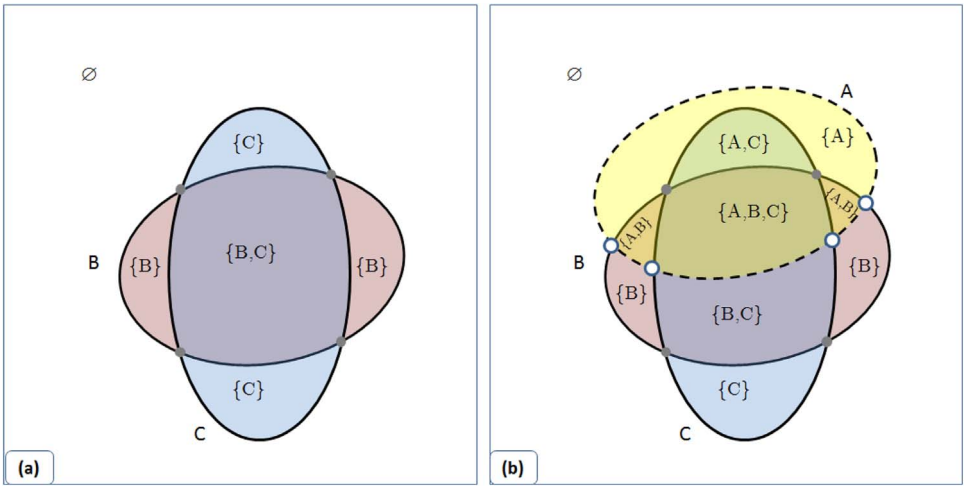

Fig. 8. An example of contour addition: (a) A non well-formed diagram $d = \langle \{B, C\}, \{\varnothing, \{B\}, \{C\}, \{B, C\}\}\rangle$. The crossing points of d are shown with filled-in dots; (b) The crossing points of A with d (i.e. those in $Cross(A)$) are depicted as hollow dots. The set of all hollow and filled-in dots depicts the set of crossing points of $d + A$.

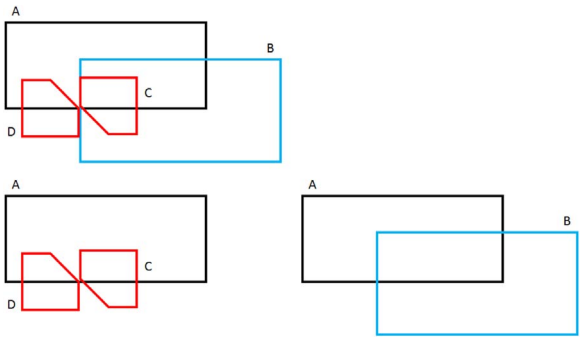

Fig. 10. The top shows a diagram and the bottom shows choices of intersecting contour sets and the associated maximal separating contour sets. On the bottom left we have $C_I = \{A, C, D\}$, and so C_J is empty, whilst on the right we have $C_I = \{A, B\}$, and so $C_J = \{C, D\}$. Neither set C_J can have further contours added to it, but the $C_I = \{A, C, D\}$ choice is the maximally sized set.

that $m(x) = i$ if there are i contours for which x is a crossing point between any pair of those contours, and i is the maximal such integer (i.e. there are exactly i non-concurrent contours that cross through x).

Definition 10. Let x be a point on an Euler Diagram d and C_I a maximal intersecting contour set for x. Then, a maximal separating contour set for C_I *at* x is a maximal set of contours $C_J \subseteq C(d) - C_I$ for which:

1. Every curve in C_J passes through x.
2. No contour c_j in C_J is locally concurrent within the set C_I.
3. No two distinct contours in C_J are locally completely concurrent at x.
4. If two distinct contours c_{j_1}, c_{j_2} in C_J are locally partially concurrent at x, then $c_{j_1} \cap c_{j_2}$ is locally concurrent within the set C_I of contours.

The *split number* of point x, denoted by $s(x)$, is $|C_J|$, the size of the set C_J. Each point x that has $s(x) > 0$ is called a *split point*.

In Definition 10, Condition (2.) for C_J prevents any of the curves in C_J from being concurrent with, but not separating from, the maximal intersecting contour set C_I, condition (3.) prevents two contours in C_J from being completely concurrent in the vicinity of x, and condition (4.) ensures that no two of the contours in C_J intersect in a common segment (i.e. are concurrent), around x, which is outside of C_I. Fig. 11 shows a complex Euler diagram d, in which the point x under consideration is indicated by a grey dot, and Fig. 12 illustrates the sets C_I for d and the possible sets C_J for each C_I. We see that $s(x) = 2$.

3. Computing the abstraction of Euler diagrams

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

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

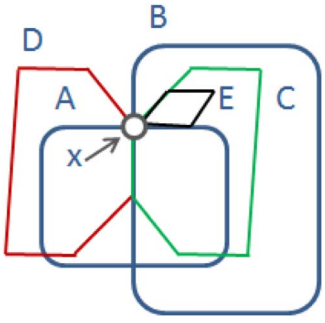

Fig. 11. A complex Euler diagram with 5 curves, illustrating the concepts in Definition 10. The point x under consideration is highlighted with a grey dot. We have $m(x) = 2$ and $s(x) = 2$ (see Fig. 12).

In [10,13] the *single marked point approach* (SMPA), described below, was presented, computing the abstraction for well-formed Euler diagrams, following an online approach where diagrams are viewed as a sequence of contour additions and removals. Fig. 13 shows the online execution of 4 operations (addition or removal of a contour) starting from an empty diagram.

In this section, 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 [13] that the set of zones split by A, due to the addition of a contour A to (or its removal from) a given well-formed Euler diagram d, can be computed using the following observation; Fig. 14 shows a schematic diagram illustrating the observation.

Observation 1. Let d be a well-formed 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 $x_{i+1 \bmod m}$).

For the well-formed diagram case of [13], 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 outer 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. In particular, the set of zones of d that are split by A are computed by: (a) choosing a point p on A and checking if p is in the interior or exterior of each curve of d to compute the zone descriptor of the initial zone; (b) making use of Observation 1 to compute the remaining zone descriptors of the split zones. 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 Fig. 15: 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 Fig. 15). 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 (Observation 1 enables the discovery of such split zones, as above); (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. Fig. 15 (c) illustrates a generalisation of the case of [13] 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.

However, the management of marked points (taking one for each minimal region) for non-well-formed diagrams (relaxing WF3) raises some tricky problems such as: if a zone z becomes split upon contour addition or removal, how can one efficiently find a marked point for each of the minimal regions that comprise z? For example, Fig. 16

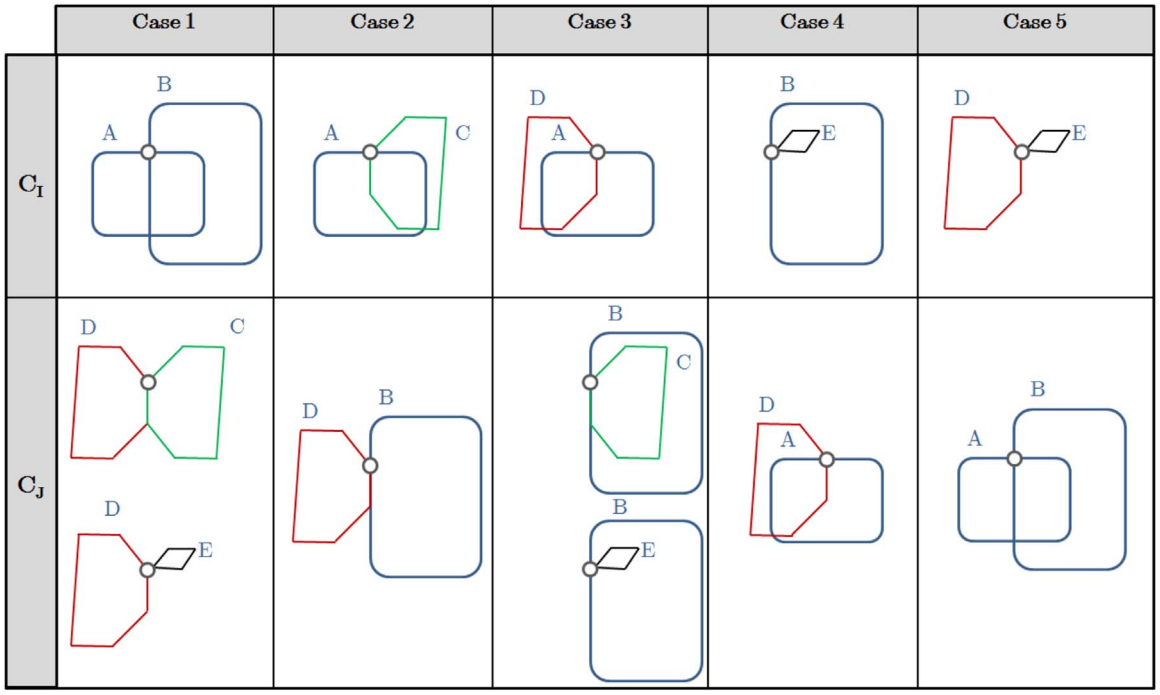

Fig. 12. The top row presents the possible choices for C_I for diagram d in Fig. 11. The bottom row shows the corresponding choices for C_J for diagram d, given C_I; for two of the five cases there is a choice of two possibilities for C_J, shown vertically above one another.

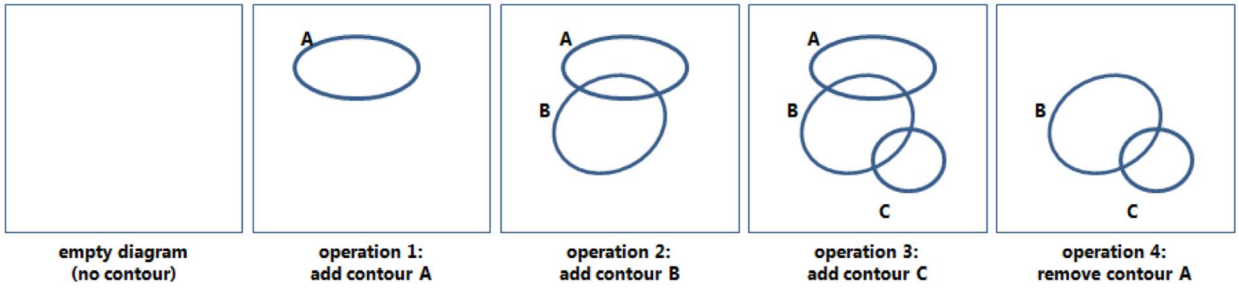

Fig. 13. A sequence of addition and removal operations on a diagram.

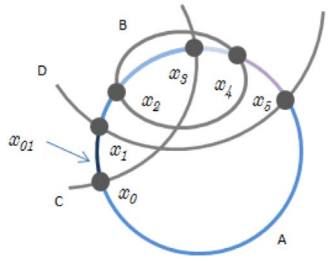

Fig. 14. A schematic diagram which illustrates Observation 1. The addition of A generates six new crossing points shown with small blobs. The point x_{01} is an arbitrary point of the arc (x_0, x_1) used to compute an initial zone descriptor for the zone split by arc (x_0, x_1), whilst subsequent zone descriptors are computed using Observation 1. The arcs of the contour A are depicted with different shades of grey in order to distinguish them.

shows two parallel examples which adopt the SMPA (using the algorithm of [13]) in which only the order of contour addition 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 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 $\{B\}$ being 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 (e.g. efficient) way of discriminating between the case on the left from the case on the right (i.e. of deciding if the two hollowed points mark the same minimal region or not). By analysing only the relationship between a single marked point and the contours it is possible to discriminate between zones but not between minimal regions.

We avoid such problem by adopting the MMPA in which each zone is associated with a set of marked points (which are primarily the set of crossing points laying on its boundary, but possibly with some extra marked points to deal with special cases). 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 Fig. 17: 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 Fig. 17, each marked point marks one, two or four zones.

In the following we define procedures for contour addition and contour removal, which satisfy:

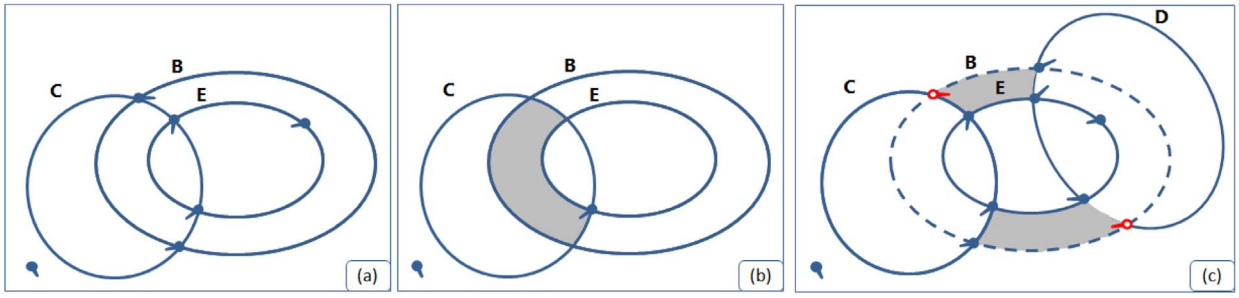

Fig. 15. Single marked point approach (SMPA): in (a) each zone of a well-formed Euler diagram is marked by a single point; (b) shows in grey the zone $\{B, C\}$ and its marked point; in (c) a non well-formed Euler diagram (WF3 relaxed) with a zone $\{B\}$, shown in grey, which consists of two minimal regions, therefore requiring at least two marked points.

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

(i) *If $A \notin C$, then the procedure* **ContourAddition**(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 $B \in C$, then the procedure* **DeleteContour**(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*:

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 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 $O(|\mathcal{Z}| + |C| + \alpha(d')\log\alpha(d'))$, where $\alpha(d')$ denotes the number of crossing points of d'.*

3.1. The algorithms

The following algorithms are presented without reference to WF conditions to avoid repetition of material. Initially, we consider the key

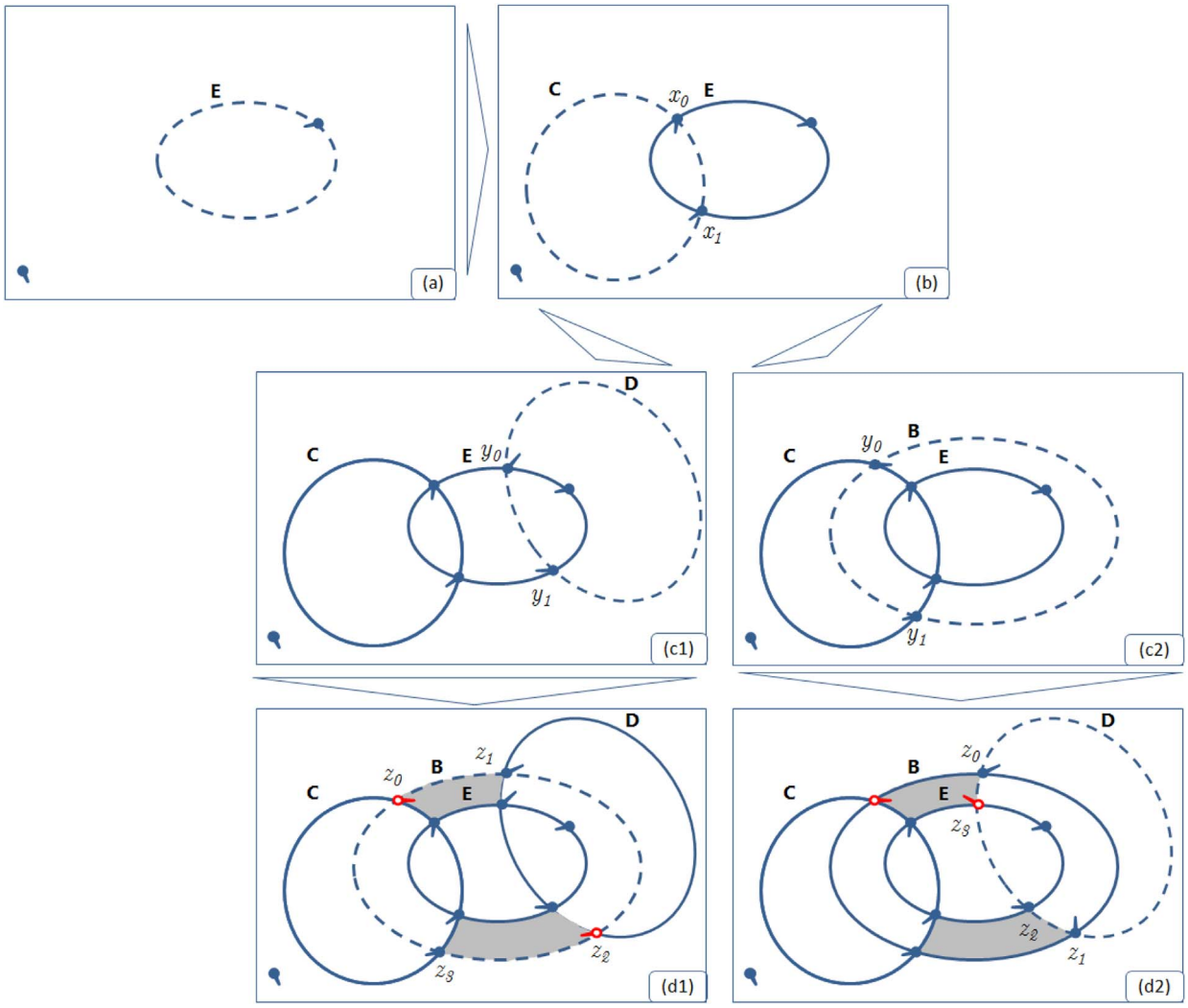

Fig. 16. The influence of the order of contour addition on the marked points/minimal region association, using the algorithms of [13]. The dotted contour is the one that is going to be added to the current diagram. The diagram d_1 has marked points correctly allocated to the minimal regions of the diagram, whilst d_2 does not.

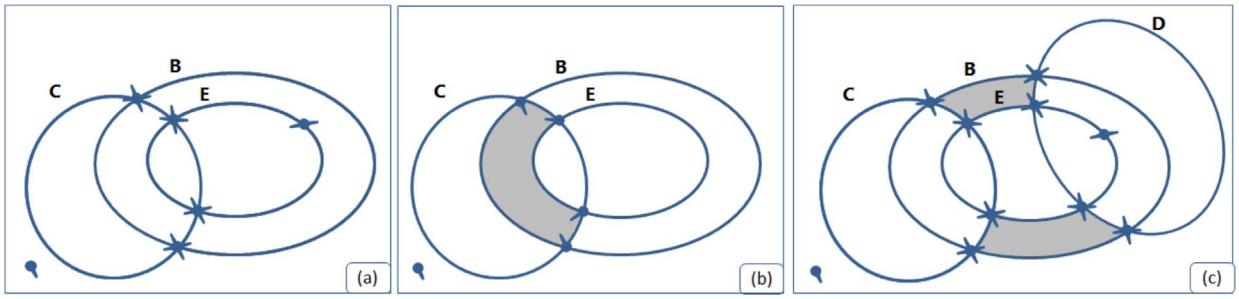

Fig. 17. Multiple marked point approach (MMPA): in (a) each zone is marked by points including all of the crossing points belonging to its boundary; (b) shows, in grey, the zone $\{B, C\}$ and its marked points; (c) a non well-formed Euler diagram (WF3 relaxed) with a zone $\{B\}$, shown in grey, comprised of two minimal regions, utilising eight marked points.

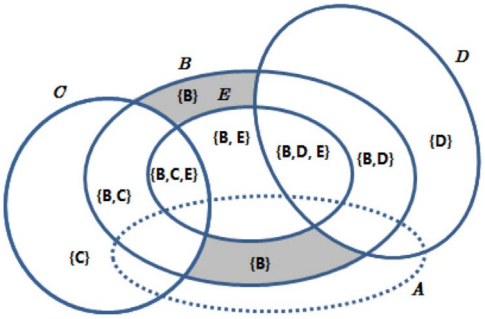

Fig. 18. The addition of contour A splits eight minimal regions determined by the eight arcs that comprise A. However, it splits nine zones, eight of which are the distinct zones containing the eight minimal regions that are split. The ninth zone $\{B\}$ is split, without splitting any of its constituent minimal regions, since one of its minimal regions is covered by A but the other is not.

case in which they are applicable to Euler diagrams with WF3 relaxed (but WF2 and WF1 enforced), but subsequently, in Sections 3.3 and 3.4 we indicate the alternations and insertions to these algorithms that are required for the extension to permit the relaxation of WF2 and WF1. This incremental approach to relaxation is intended to improve readability.

Before we describe the algorithms for contour addition and removal, we present two auxiliary algorithms, **ComputeContourRels** (which computes the relationship of a contour with the other contours in a diagram, and updates the marked point set) and **ComputeSplitRegions** (which computes the zone descriptors of the split zones).

Definition 11. Let $d = \langle C, \mathcal{Z} \rangle$ be a concrete Euler diagram and A a contour which is not in C. Let

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

For example, Fig. 8(a) shows diagram d and Fig. 8(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 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 $ext(A)$.

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). In this paper, we assume that the properties (1.) and (3.) can be calculated in $O(1)$, while (2.) can be calculated in $O(|Cross(A, B)|)$. Compute A's relationship with d and update the crossing points of d Algorithm 1: (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 according to the whether contour addition (i.e., when $A \notin C$) or contour removal (i.e., when $A \in C$) is considered.

In detail, for each contour E in $C - \{A\}$ (line 3), the algorithm checks if E belongs to $Cont(A)$(line 4), or $Over(A)$ (line 7). Then, for each contour in $Over(A)$, the collection $Cross(A)$ is updated (lines 9–10). This gives the contour relationships.

The set $Cross(d')$ is updated using Algorithm 2. Each crossing point keeps track of the two contours involved in the intersection via the property *.contours*. If A is a new contour ($A \notin C$), then this property is coherently updated (line 3), and each new crossing point is added to the set of all the crossing points in d (line 4). If A was already present in d (contour removal), then each crossing point x is removed from $Cross(d')$ since x ceases to be a crossing point (lines 6–7).

We will refer to Fig. 18 to assist with the explanation of the algorithms. Consider the addition of the dashed contour A to the diagram in Fig. 18 without A. After the execution of Algorithm 1 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 1. ComputeContourRels (d, A)

Input: An Euler Diagram $d = \langle C, \mathcal{Z} \rangle$ and a contour A.
Output: Computes the sets $Cont(A)$, $Over(A)$, $Cross(A)$and $Cross(d')$. Each crossing point in $Cross(d')$ keeps a reference to the contours which pass through it.

```
1:  Cont(A):=Over(A):=Cross(A):=∅
2:  Cross(d'):=Cross(d)
3:  forall E ∈ C − {A} do
4:      if E properly contains A then
5:          Cont(A):=Cont(A) ∪ {E}
6:      else
7:          if A and E properly overlap then
8:              Over(A):=Over(A) ∪ {E}
9:              let CP = {x_0, x_1, ..., x_{r−1}}
10:                 be the set of crossing points between A and E
11:             Cross(A):=Cross(A) ∪ CP
                UpdateCrossingPoints(d, A, E, CP, Cross(d'))
12: return (Cont(A), Over(A), Cross(A), Cross(d'))
```

algorithm 2. UpdateCrossingPoints $(d, A, E, CP, Cross(d'))$

Input: An Euler Diagram $d = \langle C, \mathcal{Z} \rangle$, two contours A and E, CP a set of crossing point between A and E and $Cross(d')$ the current set of all of the intersection points in d' to be updated.
Output: The updated set $Cross(d')$.

1: **if** $A \notin C$ **then** //Contour Addition
2: | **forall** $x \in CP$ **do**
3: | | x. contours:={A, E}
4: | | $Cross(d') := Cross(d') \cup \{x\}$
5: **else** //Contour Removal
6: | **forall** $x \in CP$ **do**
7: | | $Cross(d') := Cross(d') - \{x\}$

Compute the regions split by A. Algorithm 3 uses the sets output by Algorithm 1 and calculates the collection of zone descriptors for the zones that contain a minimal region of $d - A$ split by A. This algorithm is used for both contour addition (i.e., when $A \notin C$) or for contour removal (i.e., when $A \in C$), where we adopt our usual convention: if $A \in C$ then we refer to the zones of $d - A$ that are split by the addition of A, whilst if $A \notin C$ then we refer to the zones of d that are split by the addition of 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 (line 2); see Fig. 14 for an example. The property *.zone* of a crossing point x_i, contains the zone descriptor of the zone split by the arc $(x_i, x_{i+1 \bmod m})$. The region that is split by the first arc (x_0, x_1) is determined by taking the set of contours that properly contain A (line 3) and then adding the contours that properly overlap with A and which contain the arc (lines 5–7). The property *.points* describes the set of points that mark a zone, and both ends of the arc (x_0, x_1) mark zone $x_0.zone$ (line 8). The zone $x_0.zone$ is then recorded as a split zone (line 9).

Each successive region that is split is calculated by computing the difference with the previously computed region using Algorithm 4 (lines 1 – 5); this idea was presented in Observation 1. After computing each split zone, the crossing points of A with the contours that meet A (i.e. those that cross A at the ends of the relevant arc) are added to the collection of marked points (denoted with the property *.points*) for that zone (line 6). The collection of split zones is updated coherently (line 12 of Algorithm 3).

algorithm 3. ComputeSplitRegions $(d, A, Cont(A), Over(A), Cross(A))$

Input: An Euler Diagram $d = \langle C, \mathcal{Z} \rangle$, a contour A, and sets $Cont(A)$, $Over(A)$ and $Cross(A)$.
Output: Z_s, the collection of zone descriptors of the zones having a minimal region split by A.

1: $Z_s := \emptyset$
2: Sort points in $Cross(A)$along the contour and let $(x_0, x_1, \ldots, x_{m-1})$ be the sorting
3: $x_0.zone := Cont(A)$
4: x_{01}:= any point on the arc (x_0, x_1)
5: **forall** $D \in Over(A)$ **do** //Computing the zone descriptor for the region split by the arc (x_0, x_1)
6: | **if** $x_{01} \in int(D)$ **then**
7: | | $x_0.zone := x_0.zone \cup \{D\}$
8: $x_0.zone.points := x_0.zone.points \cup \{x_0, x_1\}$
9: $Z_s := Z_s \cup \{x_0.zone\}$
10: **forall** $i = 1, 2, \ldots, m-1$ **do** //Computing the zone descriptor for the regions split by the arcs (x_1, x_2), ..., (x_{m-1}, x_0)
11: | $x_i.zone$:=**ComputeNextZone**$(x_{i-1}.zone, x_i)$
12: | $Z_s := Z_s \cup \{x_i.zone\}$
13: **return** Z_s

algorithm 4. ComputeNextZone $(x_{i-1}.zone, x_i)$

Input: The zone associated with the point x_{i-1} and the point x_i.
Output: The zone associated with the point x_i.

1: $C := x_i$. contours − {A} //C is the contour that with A generates x_i
2: **if** $C \in x_{i-1}.zone$ **then**
3: | $x_i.zone := x_{i-1}.zone - \{C\}$
4: **else**
5: | $x_i.zone := x_{i-1}.zone \cup \{C\}$
6: $x_i.zone.points := x_i.zone.points \cup \{x_i, x_{i+1 \bmod m}\}$
7: **return** $x_i.zone$

Contour addition. Algorithm 5 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 the contours in $Cont(A)$(in Fig. 6 (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. 18 splits several zones, for example). Algorithm 5 computes the split zones in two steps: (i) the split zones which contain a split minimal region are computed by Algorithm 3; (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. That is, if none of the marked points for a minimal region are in the exterior of A then that region is covered by A and a zone is covered (respectively split) by A if all of its minimal regions are covered by A (respectively, some but not all of its minimal regions are covered).

For instance, in Fig. 18, the zones having a minimal region split by A are $\{\emptyset, \{C\}, \{B, C\}, \{B, C, E\}, \{B, E\}, \{B, D, E\}, \{B, D\}, \{D\}\}$, whilst 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.

algorithm 5. ContourAddition (d,A).

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

1: $(Cont(A), Over(A), Cross(A), Cross(d'))$:=**ComputeContourRels**$(d, A)$
2: **if** $Over(A) = \emptyset$ **then** //A does not properly overlap any contour present in C
3: | $s := Cont(A)$ //s is the zone split by A
4: | $Z_s := \{s\}$ //Z_s is the set of zones having a minimal region split
5: | $s.points$:=any point in A //the marked point for s
6: **else**
7: | Z_s:=**ComputeSplitRegions**$(d, A, Cont(A), Over(A), Cross(A))$
8: $\mathcal{Z}' := \mathcal{Z}$
9: **forall** $z \in Z_s$ **do**
10: | $s := z$ //s is the old zone
11: | $n := z \cup \{A\}$ //n is the new zone
12: | $M_s := M_n := M_A := \emptyset$
13: | **forall** $x \in s.points$ **do**
14: | | **switch do**
15: | | | **case** $x \in int(A)$ //the point x marks n
16: | | | | $M_n := M_n \cup \{x\}$
17: | | | **case** $x \in ext(A)$ //the point x marks s
18: | | | | $M_s := M_s \cup \{x\}$
19: | | | **case** $x \in A$ //the point x marks both s and n
20: | | | | $M_A := M_A \cup \{x\}$
21: | $s.points := M_s \cup M_A$
22: | $n.points := M_n \cup M_A$
23: | $\mathcal{Z}' := \mathcal{Z}' \cup \{n\}$ //The new zone n is added to the collection of zones of the diagram
24: **forall** $z \in \mathcal{Z} - Z_s$ **do**

```
25:   M_int:=M_ext:=∅        // M_int and M_ext record the marked points of z
26:                          that are in the interior and the exterior of A, respectively
27:   forall x ∈ z. points do
28:     if x ∈ int(A) then
29:       M_int:=M_int ∪ {x} else
30:       M_ext:=M_ext ∪ {x}
31:   if M_ext = ∅ then    // if z is covered by A then z should be removed
32:     Z':=Z' − {z}
33:   else
34:     z. points:=M_ext
35:
36:   if M_int ≠ ∅ then    // if z is split or covered by A then a
37:                          new region should be added
38:     n:=z ∪ {A}
        n. points:=M_int
        Z':=Z' ∪ {n}
39: return Z'
```

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, described by *Cont*(*A*), and any point on *A* can be chosen as the marked point for both the old zone and the *new zone* of *d*+*A*, which are described by *Cont*(*A*)and $Cont(A) \cup \{A\}$ 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', whilst the variable *n* refers to the new zone that is created in d' from *s* but 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).

The Algorithm **DeleteContour** to update the zone descriptors of a diagram *d* upon the removal of a contour *B* from *d*, and its description, is presented in Section 3.2.

3.2. The contour removal algorithm

Algorithm 6 updates the zone descriptors of a diagram *d* upon the removal of a contour *B* from *d*. There are two cases to consider. Firstly, if $Over(B) = \emptyset$ then *B* does not cross any other contour in *C*. Thus, *B* splits only the zone described by *Cont*(*B*). However, if $Over(B) \neq \emptyset$ then *B* splits several zones, and Algorithm 6 computes these split zones in two steps: (i) the zones of $d - B$ having a minimal region split by *B* are computed by Algorithm 3; (ii) the split zones of $d - B$ which do not have any split minimal regions are computed, together with the covered zones, by analysing the relationship between the collection of marked points and the contour *B* (similar to the contour addition case).

In detail, if *Over*(*B*)is empty then we have exactly one region split, which is contained within the zone described by *Cont*(*B*); we record the zone descriptor in the variable *s* (line 3) and add *s* to the set of split zones Z_s (line 4). Then, for each zone containing a split minimal region, *r*, the new zone *n* (i.e. the zone containing *r* whose descriptor contains *B*; the terminology is chose to be consistent with the view of the contour addition of *B* to $d - B$) is removed, whilst the old zone *s* (i.e. the zone containing *r* whose descriptor does not contain *B*) is retained. The set of marked points is also coherently updated so that *s* retains all of the marked points of *s* and *n* which remain crossing points in d' (lines 8–15).

Subsequently (lines 16–24), the zones *z* of *d* which do not have any minimal regions split by *B* (i.e. by the addition of *B* to $d - B$) are checked to see if they are covered or split zones of $d - B$. This is performed by checking if *B* belongs to the zone descriptor of *z*. If so, then *z* is removed from $\mathcal{Z}$ (line 18). Moreover, if $z - \{B\} \notin \mathcal{Z}$ then the zone *z* is covered by *B*, and so the zone $z - \{B\}$ is added to $\mathcal{Z}$ (line 21); otherwise the zone *z* is split by *B*. In both cases the set of marked points are updated coherently (lines 22, 24).

algorithm 6. DeleteContour (*d*,*B*).

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

```
1:  (Cont(B), Over(B), Cross(B), Cross(d')):=ComputeContourRels(d, B)
2:  if Over(B) = ∅ then      //B does not properly overlap any
                               contour in C − {B}
3:    s:=Cont(B)
4:    Z_s:={s}
5:  else
6:    Z_s:=ComputeSplitRegions(d, B, Cont(B), Over(B), Cross(B))
7:  Z':=Z
8:: for all z ∈ Z_s do
9:    s:=z
10:   n:=z ∪ {B}
11:   s. points:=s. points ∪ n. points
12:   forall x ∈ s. points do
13:
14:     if x ∉ Cross(d') then      // x does not remain a
15:                                  crossing point
        s. points:=s. points − x
      Z':=Z' − {n}
16: forall z ∈ Z − Z_s do
17:   if {B} ∈ z then      // z is covered or split by B
18:
19:     Z':=Z' − {z}
20:     s:=z − {B}
21:     if s ∉ Z then      // z is covered by B
22:       Z':=Z' ∪ {s}     // s is added to Z'
23:       s. points:=z. points
24:     else               // z is split by B
        s. points:=s. points ∪ z. points      // update marked points
25: return Z'
```

3.3. Relaxing the well-formedness condition 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. 19 presents a schematic diagram, similar to that shown in Fig. 14, 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).

algorithm 7. UpdateCrossingPoints (d, A, E, CP, $Cross(d')$) for WF2.

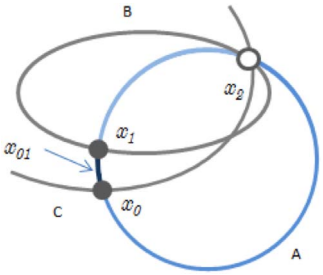

Fig. 19. A schematic diagram illustrating the need for Observation 2 when WF2 is relaxed (c.f. Fig. 14). The addition of contour A creates two new crossing points of multiplicity 2, x_0 and x_1 (shown as filled dots), and creates one intersection point, x_2, having multiplicity 3 (shown as a hollow dot). Suppose that (x_0, x_1) splits zone $\{C\}$. Then (x_1, x_2) splits zone $\{B, C\}$, and (x_2, x_0) splits the zone $\emptyset$ (which differs from $\{B, C\}$ by the absence of the two contours, B and C). The arcs of the contour A are shown using different gradients of grey in order to distinguish them.

1: **if** $A \notin C$ **then** //Contour Addition
2: **forall** $x \in CP$ **do**
3: **if** $x \in Cross(d')$ **then** // x's multiplicity > 2
4: x. contours:=x. contours $\cup$ {A}
5: **else** // x's multiplicity $= 2$
6: x. contours:={A, E}
7: $Cross(d')$:=$Cross(d') \cup \{x\}$
8: **else** //Contour Removal
9: **forall** $x \in X_E$ **do**
10: x. contours :=x. contours $-$ {A}
11: **if** |x. contours| < 2 **then** //x ceases to be an crossing point
12: $Cross(d')$:=$Cross(d') - \{x\}$

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; these are the contours that intersect with A comprising the crossing point $x_{i+1 \bmod m}$.

The algorithms in Section 3 are altered to deal with the relaxation of WF2 as follows. Firstly, in Algorithm **UpdateCrossingPoints** each crossing point kept track of the two contours generating it, via the property *.contours*. We now require that each crossing point keeps track of every contour that passes through it. The algorithm **UpdateCrossingPoints** for WF2 below replaces Algorithm **UpdateCrossingPoints** in order to deal with the relaxation of WF2. In detail, in **UpdateCrossingPoints** for WF2, if A is a new contour ($A \notin C$), then the *.contours* property is coherently updated, depending upon whether each intersection point is already present in the diagram (lines 3–4) or has been created by the addition of A (lines 6 – 7). If A is an existing contour in d (for contour removal), we first remove A from x.contours (line 10) and if the size of the remaining set of contours that pass through x is smaller than 2 (line 11), then x is removed from $Cross(d)$ since x ceases to be a crossing point.

Secondly, we need to update Algorithm **ComputeNextZone**, replacing it with Algorithm 8, called **ComputeNextZone** for WF2. Region descriptions are computed (utilising Observation 2) by computing the change from the previously computed region description (see Fig. 19).

algorithm 8. ComputeNextZone $(x_{i-1}.\ zone, x_i)$ for WF2

1: $\mathcal{V}$:=x_i. contours //$\mathcal{V}$ is the collection of contours which pass through x_i
2: **forall** $C \in \mathcal{V} - \{A\}$ **do** // for each contour in $C \in \mathcal{V} - \{A\}$

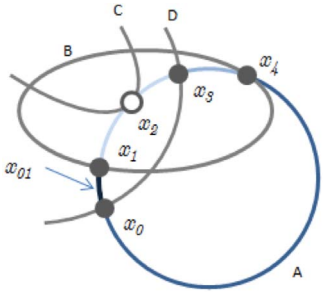

Fig. 20. The addition of A, yielding a diagram with WF1a relaxed. We have four transverse crossing points (shown as filled dots) and one tangential intersection point (shown as a hollow dot). Suppose that (x_0, x_1) splits zone $\{D\}$. Then both of (x_1, x_2) and (x_2, x_3) split zone $\{B, D\}$ since the intersection point x_2 does not affect the zone which is split (by the addition of A). Then (x_3, x_4) splits the zone $\{B\}$ and (x_4, x_4) splits the zone $\emptyset$, as usual. The arcs of the contour A are shown using different gradients of grey in order to distinguish them.

3: **if** $C \in x_{i-1}.\ zone$ **then**
4: $x_i.\ zone$:=$x_{i-1}.\ zone - \{C\}$
5: **else**
6: $x_i.\ zone$:=$x_{i-1}.\ zone \cup \{C\}$
7: $x_i.\ zone.\ points$:=$x_i.\ zone.\ points \cup \{x_i, x_{i+1 \bmod m}\}$
8: **return** $x_i.\ zone$

The algorithms **ContourAddition** and **DeleteContour** require no further changes in order to deal with the relaxation of WF2.

3.4. Relaxing the well-formedness condition WF1

Firstly, we relax WF1a. Since tangential intersection points do not affect the zones which are split by the corresponding arc (see Fig. 20), we adapt the algorithm to deal with tangential intersections by simply ignoring them. Thus the property *.contours* associated with each crossing point should keep track of every contour that it crosses *transversely*. In line 2 of algorithm **UpdateCrossingPoints** the **forall** should range over only the crossing points that cross A transversally. For this part, we assume that, given two contours A and B, we can quickly find the type of each intersection point (i.e. if it is a tangential intersection or a transverse crossing). The rest of the algorithms remain unchanged.

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.e., in $O(1)$ time):

1. check whether they are locally non-concurrent at a point x or not.
2. check if a concurrent arc at a point x is tangential or transversal (see Definition 8, and Fig. 21 (left), (right) for examples of tangential and transverse split points respectively).
3. find the split points (the points where two contours that meet in a concurrent arc separate).

The split points play essentially the same role as the crossing points within the extended algorithms: they are 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 Fig. 21 left), and so we treat such points in the same manner as tangential intersections; we refer to them as *tangential split points*. In the algorithms, the property *.contours* associated to split points will not contain any contour that forms a concurrent arc with A and separates at the split point. For a transversal concurrent arc, the first split point that we encounter as we traverse the contour A does not affect the zone

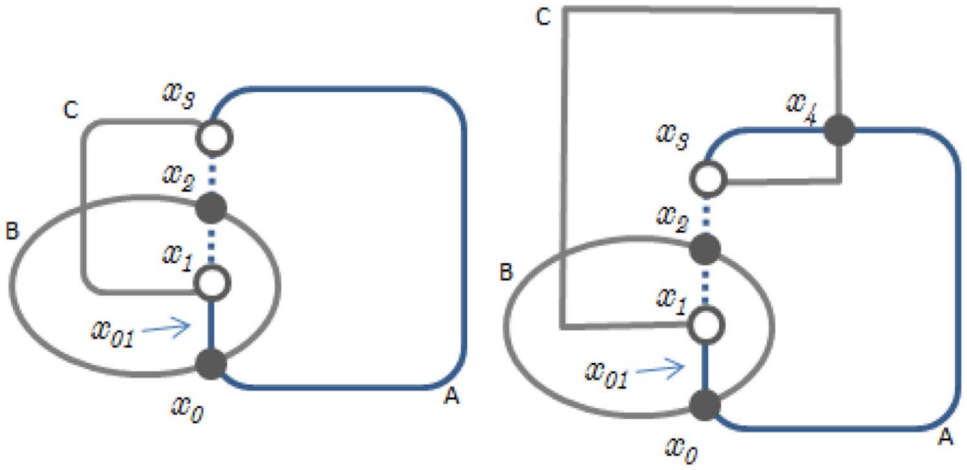

Fig. 21. The addition of A, yielding diagrams with WF1b relaxed. (left) The contour A creates two transverse crossing points, x_0 and x_2, and one tangential concurrent arc between points x_1 and x_3 (shown as hollow dots). The points x_1 or x_3 do not affect the splitting of any zones. (right) The addition 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). The first split point that we meet as we traverse the contour A from point x_0 is x_1, and it does not affect the splitting of any zone (i.e., the split zone is $\{B\}$ for both the arc before and the arc after the point x_1); the second split point encountered, x_3, does affect the zone splitting (i.e., the split zone is $\varnothing$for the arc before x_3 and $\{C\}$ for the arc after it).

which is split, whilst the corresponding second split point does affect the zone which is split (see Fig. 21 right). Therefore, the first point will be treated as a tangential intersection while the second one will be treated as a transverse crossing; the first split point is tangential, whilst the other is a *transversal split point*). If a point x is a crossing-split point (see Section 5), then the property *.contours* associated to x will keep track of every contour that *transversely* passes through x, as well as every contour which creates a transversal split point at x.

3.5. Timing

We provide complexity analysis for the MMPA for Euler diagrams, and indicate how the approach compares in terms of efficiency with alternative methods.

The invocation of the procedure **ComputeContourRels** analyses the relationship between A and each contour in C, and so it takes time $O(|C|)$. Moreover, for each contour $E \in C$, the collection CP of crossing points of E with A, is compared with $Cross(d)$, the set of all of the crossing points of the diagram prior to the addition of A, in such a way as to: (i) compute $Cross(d')$; (ii) update, for each point in $Cross(d')$, the collection of contours which pass through it. Therefore, we check whether each point in $Cross(A)$is already present in $Cross(d)$. Since both $|Cross(A)|$ and $\alpha(d - A)$ are bounded above by $\alpha = \alpha(d)$, we can perform the check, for all of the points in $Cross(A)$, in $\alpha \log \alpha$ time. Therefore, the whole procedure takes time $O(|C| + \alpha \log \alpha)$, assuming that we are using an efficient data structure to maintain the intersection points and the contours to which they belong.

Then, if there are no intersection points (i.e., if $Over(A) = \varnothing$), the split region is computed within $O(1)$ time. Otherwise, if the contour A creates intersection points, then the procedure **ComputeSplitRegions** computes the set, Z_s, of zones having a split region and, for each such zone it updates the set of marked points. The sorting of the intersection points, between A and d, requires at most $O(\alpha \log \alpha)$ time. The computation of the first zone descriptor requires $O(|C|)$ steps, whilst each subsequent zone descriptor is computed in $O(1)$ time. Hence, the procedure **ComputeSplitRegions** requires $O(|C| + \alpha \log \alpha)$ 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. Notice that even if the same point can be checked against A several times in the switches (lines 15, 17, 19 and 27), the result of the comparison can be cached. Assuming that we have $O(\alpha)$ locations available to keep the result of the comparisons for caching purposes, we can perform the two **forall** cycles in $O(|\mathcal{Z}| + \alpha)$ steps. The analysis is similar for the contour removal case. Collectively, the algorithms **ContourAddition** and **DeleteContour** operate within time $O(|\mathcal{Z}| + |C| + \alpha \log \alpha)$.

We note that the output of these algorithms are a set of zone descriptions of cardinality $\mathcal{Z}$. Hence the timing of any abstraction algorithm is at least $\Omega(\mathcal{Z})$, which means that if $O(|C| + \alpha \log \alpha)$ is dominated by $O(\mathcal{Z})$, then our solution is optimal. Our analysis is based on the assumptions (1.)–(3.) of Section 3.1 and the assumption (1.)–(3.) of Section 3.4. If this is not the case, then the overall complexity will be worse, but it will still be dominated by $O(2^{|C|})$, which represents the complexity of a standard graph-based approach. Considering that: (i) in the average case, the number of zones is much smaller than $O(2^{|C|})$; and (ii) our algorithm follows an inductive approach (which greatly reduces the number of comparisons required), the performance of the algorithms is very efficient in the field (see Section 6).

4. Extensions of the algorithms

To simplify the main set-up and assist readability, we integrated two conditions into the definition of a concrete Euler diagram that have been considered as well-formedness conditions in some of the literature (i.e. uniqueness of contour labels and simplicity of curves). Now, we extend the definitions and algorithms in order to deal with these cases. In Definition 12, we permit non-unique contour labels and allow multiple labels to be assigned to a single contour (this can be interpreted as placing multiple singly labelled curves on top of each other).

Definition 12. A *generalised* concrete Euler diagram is a pair (d, η), where $d = \langle C, \mathcal{Z} \rangle$ is a concrete Euler diagram, and η is a (total) labelling function $C \to \mathcal{P}(L)$, with $\mathcal{L}$ a set of labels. The label set of (d, η), denoted by $\eta(d)$, is $\bigcup_{c \in C} \eta(c)$. We say that a label l is a k- label of a zone descriptor X_z if $|\{c_i \in X_z : l \in \eta(c_i)\}| = k$.

Since we utilised contour identifiers instead of labels in the Euler diagram set-up (with an obvious correspondence for uniquely labelled contours), we can consider the labelling function independently from the Euler diagram. Thus we can utilise the previous methodology and algorithms for Euler diagrams, and provide post-processing manipulation to provide an interpretation for generalised Euler diagrams. An advantage of this approach is that the interpretation function can be varied according to the situation requirements (e.g. the application domain). We present two possible interpretation functions to demonstrate this point. The interpretation of a generalised Euler diagram is determined by a choice of the meaning of the concept of *inside* for a contour label.

Definition 13. Let (d, η) be a generalised concrete Euler diagram. A

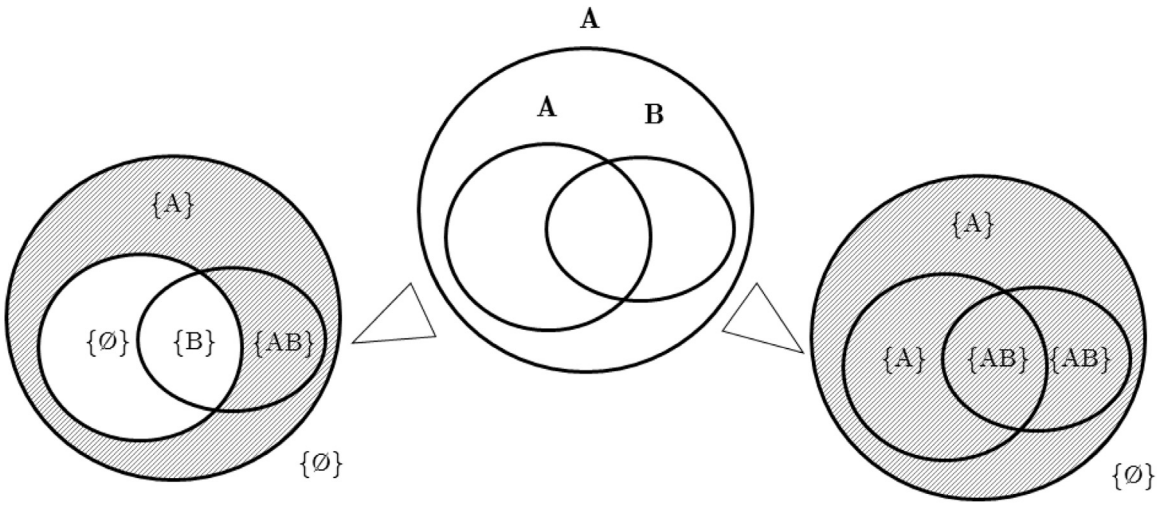

Fig. 22. A generalised Euler diagram which has different abstractions (i.e. different abstract zone sets) depending upon the interpretation chosen. Interpretation 1 (left) effectively interprets any contour label as a union of regions with holes, whilst Interpretation 2 (right) effectively interprets any contour label as a union of regions without holes. For example, on the left the interior of label A is the annulus formed by the two contours labelled A, whilst on the right it is the disc formed from the outer contour labelled A (shown shaded).

point $p \in \mathbb{R}^2 - C$, where C is viewed as a set of points, is *inside* label A if: .

Interpretation 1 any path from p to a point x in the unbounded region of the plane crosses an odd number of contours with label A,
Interpretation 2 it is in the interior of any contour with label A,

and is *outside* A otherwise.

This determines the abstract Euler diagram that the generalised Euler diagram represents. Fig. 22 shows an example demonstrating distinct abstractions as Euler diagram according to the different interpretations of 'inside' of a contour label.

Algorithm 9 computes the zones of the abstract Euler diagram for a generalised Euler diagram under either Interpretation 1 or Interpretation 2. In detail, this algorithm first computes the set of zone descriptors for the underlying Euler diagram d (line 2), ignoring the labelling function η. For each zone descriptor (line 3), the algorithm computes z, the abstract zone corresponding to the concrete zone according to the interpretation function. That is, label l is added to the set of labels of the abstract zone if and only if the concrete zone is inside k contours with label l for Interpretation 1 or inside any contours with label l for Interpretation 2 (lines 5 – 9). Each of these abstract zones is recorded, together with the outside zone (lines 1 and 9).

algorithm 9. ComputeZoneSetGeneralisedED1 (d)

Input A concrete generalised Euler Diagram (d, η) with $d = \langle C, \mathcal{Z} \rangle$ an Euler diagram, η a labelling function, and I an interpretation function returning interpretation i for $i \in \{1, 2\}$.
Output Computes $Z(d)$, the set of abstract zones under interpretation i.

1: $Z(d) := \{\emptyset\}$
2: Compute $\overline{Z(d)}$ the set of zones descriptors for the underlying diagram $\overline{d}$
3: **forall** $\overline{z} \in \overline{Z(d)}$ **do**
4: $\quad z := \emptyset$
5: $\quad$ **forall** $l \in \eta(d)$ **do**
6: $\quad\quad$ **if** l *is a* $k-$*label of* $X_{\overline{z}}$ **then**
7: $\quad\quad\quad$ **if** (k *is odd OR* $I(d, \eta) = 2$) **then**
8: $\quad\quad\quad\quad z := z \cup \{l\}$
9: $\quad\quad Z(d) := Z(d) \cup \{z\}$
10: **return** $(Z(d))$

Fig. 23 shows an example of the use of Interpretation 1 for generalised Euler Diagrams.

It is easy to see that use of multiple non-unique labelling ensures that any abstract diagram has a realisation, under either interpretation (see Lemma 1). Of course, the diagram constructed in the below proof is not necessarily a sensible choice from a user perspective – utilising diagrams that are 'as well-formed as possible' is likely to be preferable from a user comprehension perspective.

Lemma 1. *Any abstract Euler Diagram has a realisation as a generalised Euler Diagram* (*under either interpretation*).

Proof. Consider each zone $z = (X, Y)$ in the abstract Euler Diagram d, in turn. If $X = \{C_1, \ldots, C_n\}$, construct a curve c with label set $\{C_1, \ldots, C_n\}$ that is disjoint from the interior of all of the previous curves constructed.□.

4.1. Extension to non-simple curves

Since the use of generalised concrete Euler diagrams ensures that any abstract diagram has a realisation, the desire for the relaxation of simplicity of curve may be limited, but we permit the interpretation of diagrams that exhibit non-simple curves, where there are a finite number of self-intersection points. We provide one possible method for viewing a set of, possibly non-simple, labelled closed curves as a generalised Euler diagram, in the proof of Proposition 1. This enables the use of the previous methodology.

Proposition 1. *Let C be a set of labelled closed curves in the plane, with a finite set of self - intersection points. Then there is a generalised Euler diagram, (d, η), where $d = (C', \mathcal{Z})$, with the following properties*:

- *for each non - simple closed curve in C there is a set of simple closed curves in C', determining the same set of points in the plane,*
- *each simple closed curve in C appears in C',*
- *no other curves appears in C',*
- *for each curve in C' the total function η assigns the label for the corresponding curve in C.*

Proof. Suppose that A is the label of a non-simple curve c with a finite set of self-intersections points. Then the complement of c in $\mathbb{R}^2$ is a union of finite set of open, bounded regions $r_1, \ldots, r_n$ of the plane, together with an open unbounded region r. Replace the curve c by a finite set of simple closed curves, $c_1, \ldots, c_n$, each labelled A, which bound $r_1, \ldots, r_n$, respectively.

The set of points in $c_1 \cup \ldots \cup c_n$ is equal to the set of points in c.□.

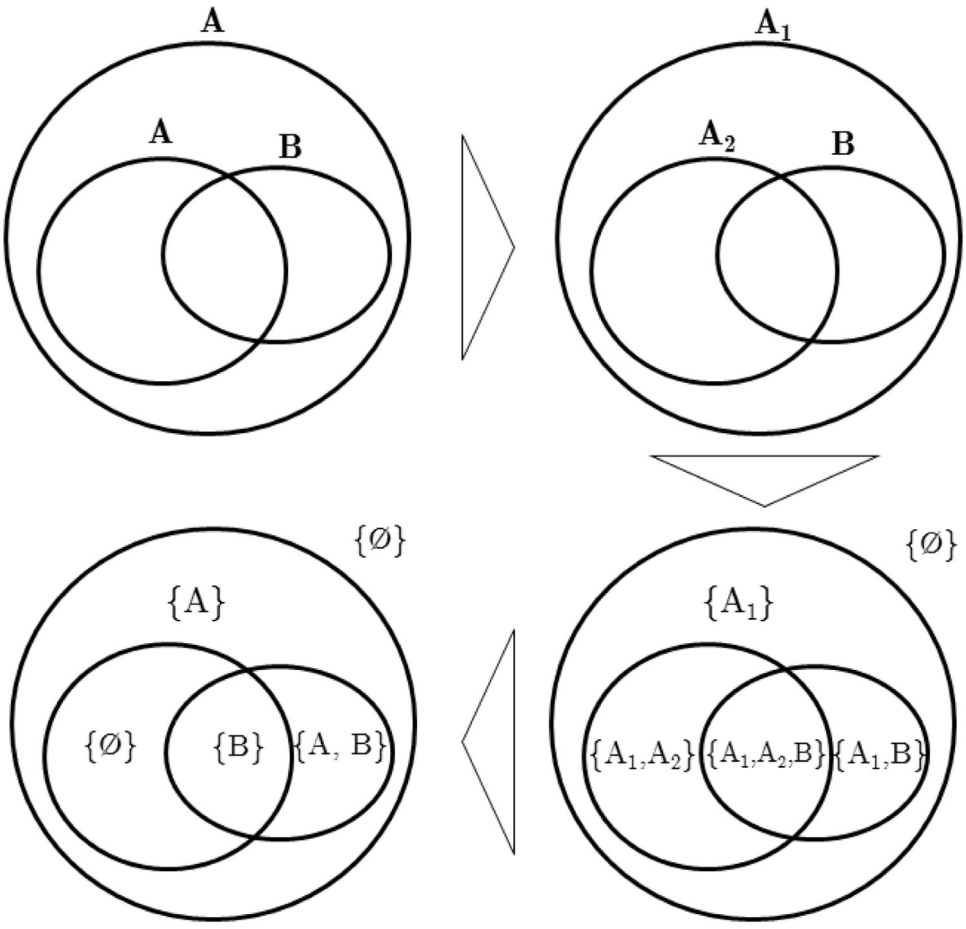

Fig. 23. An example demonstrating the application of the algorithm to interpret a generalised Euler diagram. We start with a generalised Euler diagram with three contours, two of which are labelled A and the other B. We view this as an Euler diagram, assigning unique identifiers to each of the curves; here we use A_1 and A_2 as identifiers for the contours labelled by A to aid readability (top right). This enables us to compute the zone descriptors for each concrete zone, utilising the unique identifiers (bottom right). Then, for each zone descriptor, we can check if the concrete zone is inside each label by checking if there is an odd number of contours in the zone descriptor in which the label appears, following Interpretation 1. Then one replaces the zone descriptor by the corresponding label set (bottom left). For example the zone descriptor A_1A_2 is replaced by $\varnothing$ since both A_1 and A_2 identify curves labelled by A.

5. Identifying disconnected zones

Since the algorithm to enable the computation of the abstraction of Euler diagrams proposed in [10,13] solves the abstraction problem only for well-formed diagrams, a method for identifying disconnected zones (WF3), when WF1 and WF2 are preserved, was developed. Here, we presented an evolution of the algorithm to enable the computation of the abstraction of any given Euler diagram (or generalised Euler diagram). However, it is always important to know whether or not a concrete diagram satisfies a certain well-formedness condition, whose imposition is intended to reduce errors of comprehension. Therefore, we want to be able to check if each diagram is well-formed, so that we can highlight the information of how the diagram fails the well-formedness conditions, for a user of a software tool, for instance. The preservation of some well-formed conditions (e.g., no self-intersection, unique labelling, WF1 and WF2) upon contour addition or removal is easily checked: contours must not generate self intersections; the label of the newly added contour must be new; each intersection of the new contour with existing contours should create a new intersection point which is a transverse crossing that is distinct from the existing intersection points; the number of intersection points must be finite (no concurrency). On the other hand, identifying disconnected zones, when WF1 and WF2 are not preserved, is not easy; we provide a result to enable the identification of disconnected zones in this general case.

Definition 14. Let $d = \langle C, \mathcal{Z} \rangle$ be a Euler diagram. Then, we:

- denote the number of crossing points of d with multiplicity i by $\alpha_i(d)$, and the total number of crossing points of d by $\alpha(d) = \sum_{i=2}^{n} \alpha_i(d)$, where n is the maximum multiplicity of any crossing point in d.
- denote the number of split points of d having split number j by $\beta^j(d)$, and the total number of split points of d by $\beta(d) = \sum_{j=1}^{m} \beta^j(d)$, where m is the maximum split number in d.
- call a point which is both a crossing point and a split point a *crossing-split point*. Let $\gamma_i^j(d)$ denote the number of points that are crossing points with multiplicity i and split points with split number j. Let $\gamma_i(d) = \sum_{j=1}^{m} \gamma_i^j(d)$ be the number of crossing-split points having multiplicity i, $\gamma^j(d) = \sum_{i=2}^{n} \gamma_i^j(d)$ be the number of crossing-split points having split number j, and

$$\gamma(d) = \sum_{i=2}^{n} \gamma_i(d) = \sum_{i=2}^{n} \sum_{j=1}^{m} \gamma_i^j(d) = \sum_{j=1}^{m} \sum_{i=2}^{n} \gamma_i^j(d) = \sum_{j=1}^{m} \gamma^j(d)$$

be the total number of crossing-split points of d.
- partition $C(d)$ into the connected components of d (i.e. the maximal sets of intersecting contours), and denote the number of components of d by $\delta(d)$.

In [13] a relationship between the number of zones, the number of crossing points (previously called intersection points) and the number of connected components of a well-formed diagram d was presented; this can be used to check if a diagram has disconnected zones (i.e d fails WF3).

Theorem 2 (*restated from [15]*). *Let $d = \langle C, \mathcal{Z} \rangle$ be obtained by adding or removing a contour from a well - formed Euler diagram. If*

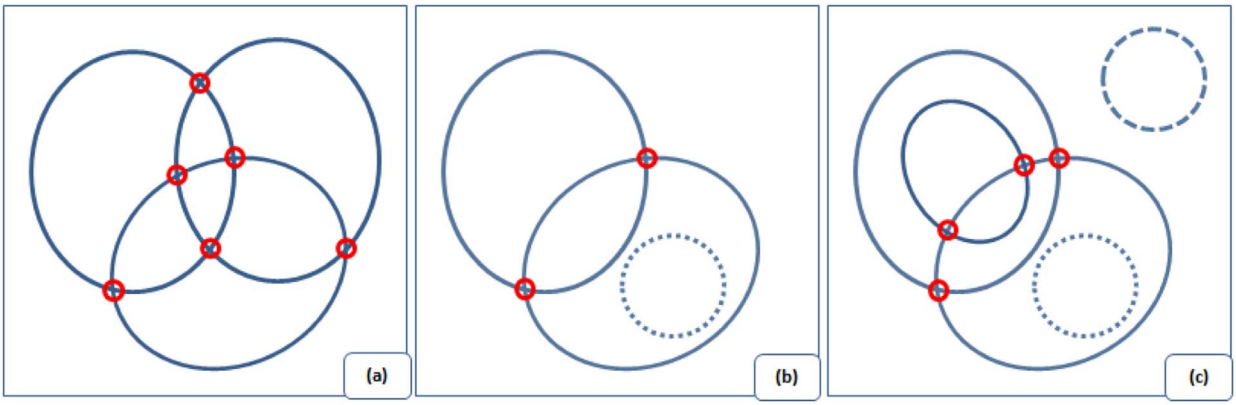

Fig. 24. Single contour components are shown dashed, and crossing points are highlighted: (a) 6 crossing points and 1 connected component, so $\alpha(d) = 6$, $\delta(d) = 1$ and $|\mathcal{Z}| = 8$; (b) 2 crossing points and 2 connected components, so $\alpha(d) = 2$, $\delta(d) = 2$ and $|\mathcal{Z}| = 5$; (c) 4 crossing points and 3 connected components, so $\alpha(d) = 4$, $\delta(d) = 3$ and $|\mathcal{Z}| = 8$.

d satisfies WF1 and WF2 then d is well - formed iff $|\mathcal{Z}| = \alpha(d) + \delta(d) + 1$.

An example, with computations demonstrating the result of Theorem 2, is provided in Fig. 24.

However since, in this paper, we allow points of tangential intersection or multiple contours crossing at a point, the above result does not apply. Therefore, we provide an extended result which relates the number of minimal regions of an Euler diagram to the number of crossing points, their multiplicity, the number of split points, their split number, and the number of connected components of the diagram. This result enables the identification of diagrams passing or failing the well-formedness condition without needing to compute the associated graphs.

Lemma 2. *Let x be a split point of a diagram d. Then the vertex degree of x in G(d), the graph of the diagram, is* $2|C_I| + |C_J| = 2m(x) + s(x)$.

Proof. Firstly, suppose that x is an crossing-split point. Then the summand $2|C_I|$ arises from maximal number of curves crossing at x (i.e. the crossing number $m(x) = |C_I|$), whilst the splitting number $s(x)$ counts the number of extra distinct segments of curves that become concurrent with the contours involved in the intersection point at x.

Secondly, if x is not an crossing point then, since x is a split point by hypothesis, we have that C_I is a single contour. Therefore, $2|C_I|$ counts the two edges at x corresponding to that contour, whilst $s(x)$ counts the number of extra distinct segments of curves that separate from any concurrency with that single contour at x.□.

We make use of the following Theorem, paraphrased from [43], pages 66–67.

Theorem 3 (*Euler's Formula*). *Let G be a connected plane graph. Then* $v - e + f = 1 + k$ *where v is the number of vertices, e is the number of edges, f is the number of faces, and k is the number of components of G.*

Theorem 4. *Let* $d = \langle C, \mathcal{Z}\rangle$ *be an Euler diagram with r minimal regions. Then* $r = 1 + \delta(d) + \sum_{i=2}^{n}(i-1)\alpha_i(d) + \frac{1}{2}\sum_{j=1}^{m} j\beta^j(d)$.

Proof. We partition the set of vertices of $G(d)$ into crossing points, split points, and crossing-split points (where these are strict terms) and vertices identifying single contour components. That is, we partition $V(G(d))$ into V_α, V_β, V_γ and V_δ where:

- V_α is the set of vertices with $v_x \in V_\alpha$ corresponding to point x of d with $m(x) > 1$ and $s(x) = 0$,
- V_β is the set of vertices with $v_x \in V_\beta$ corresponding to point x of d with $m(x) = 1$ and $s(x) > 0$,
- V_γ is the set of vertices with $v_x \in V_\gamma$ corresponding to point x of d with $m(x) > 1$ and $s(x) > 0$,
- V_δ is the set of vertices with $v_x \in V_\delta$ corresponding to point x of d with $m(x) = 1$ and $s(x) = 0$, where x belongs to a component that consists of a single contour.

Each vertex

- $v_x \in V_\alpha$ with $m(v_x) = i$ has $deg(v_x) = 2i$ (where *deg* is the degree of the vertex),
- $v_x \in V_\beta$ with $s(v_x) = j$ has $deg(v_x) = j + 2$ by Lemma 2,
- $v_x \in V_\gamma$ with $m(v_x) = i$ and $s(v_x) = j$ has $deg(v_x) = 2i + j$ by Lemma 2,

Except for the edges incident with vertices in V_δ, each edge of $G(d)$ joins distinct vertices, and so we have:

$$
\begin{aligned}
e &= \frac{1}{2}\left(\sum_{i=2}^{n} 2i(\alpha_i(d) - \gamma_i(d)) + \sum_{j=1}^{m}(j+2)(\beta^j(d) - \gamma^j(d))\right) \\
&\quad + +\frac{1}{2}\left(\sum_{i=2}^{n}\sum_{j=1}^{m}(2i+j)\gamma_i^j(d)\right) + |V_\delta| \\
&= \frac{1}{2}\left(\sum_{i=2}^{n}(2i\alpha_i(d) - 2i\gamma_i(d)) + \sum_{j=1}^{m}((j+2)\beta^j(d) - (j+2)\gamma^j(d))\right) \\
&\quad + +\frac{1}{2}\left(\sum_{i=2}^{n} 2i\gamma_i(d) + \sum_{j=1}^{m} j\gamma^j(d)\right) + |V_\delta| \\
&= \frac{1}{2}\left(\sum_{i=2}^{n} 2i\alpha_i(d) + \sum_{j=1}^{m}(j+2)\beta^j(d)\right) - \sum_{j=1}^{m}\gamma^j(d) + |V_\delta| \\
&= \sum_{i=2}^{n} i\alpha_i(d) + \frac{1}{2}\sum_{j=1}^{m}(j+2)\beta^j(d) - \sum_{j=1}^{m}\gamma^j(d) + |V_\delta|.
\end{aligned}
$$

Since d is a plane graph, by Euler's formula we have $v - e + f = 1 + k$, where: f is the number of faces in d which is equal to r, the number of minimal regions in the diagram; $v = \alpha(d) + \beta(d) - \gamma(d) + |V_\delta|$; k is the number of components of d, which is equal to $\delta(d)$. Therefore,

$$
\begin{aligned}
f &= 1 + k - v + e = 1 + \delta(d) - (\alpha(d) + \beta(d) - \gamma(d) + |V_\delta|) \\
&\quad + \sum_{i=2}^{n} i\alpha_i(d) + \frac{1}{2}\sum_{j=1}^{m}(j+2)\beta^j(d) - \sum_{j=1}^{m}\gamma^j(d) + |V_\delta| \\
&= 1 + \delta(d) + \sum_{i=2}^{n}(i-1)\alpha_i(d) + \frac{1}{2}\sum_{j=1}^{m} j\beta^j(d)
\end{aligned}
$$

as required.□

We notice that Theorem 4 generalises the result in Theorem 2. Indeed, when the diagram satisfies WF1 and WF2, there are no split points and every crossing point has multiplicity 2. Thus, one can identify and highlight occurrences of the failure of the relevant WF conditions, which can be useful in conjunction with the main algo-

rithms and well as for presentation within an interactive environment.

We can view diagram d as a graph $G(d)$ whose vertices are the crossing points and the split points of d, provided that we add an additional vertex at any point of each contour that does not intersect with any other contour in d, and whose edges are the connecting segments of the contours. For the case of connected diagrams, this was called the graph of the diagram in [8].

6. Implementation and benchmarking with ellipses

The algorithms presented in this paper were designed in generality, not placing restrictions on the types of curves used (e.g. in terms of geometric shape). In some application areas, such restrictions are imposed, and in [10], arbitrarily rotated ellipse-based Euler Diagrams were used in a visual interface for information classification. Within this application context, a Java library that hosts the data structures and the algorithms presented in this paper have been constructed. We provide a performance evaluation of the implementation in order to indicate the potential practical utility of the algorithms. However, we note that the library was implemented for demonstration purposes, without particular attention paid to performance, and an improved implementation (particularly if context specific) would be very likely to achieve higher speed.

Benchmark setting. The benchmark was designed with the intention of assessing the performance of the library in the context of an interactive application, making good use of the on-line nature of the algorithms. Each run of the test begins with the generation of a random diagram containing arbitrarily placed, rotated and scaled ellipses that will create numerous intersection points. Then an “interactive phase” is performed, simulating a user's interactions with the diagram by iteratively adding a random ellipse (random orientation, size and placement) and removing a random ellipse. This mimics three types of user interactions: adding a new ellipse, deleting an ellipse, modifying an ellipse (which can be viewed as a removal followed by an addition). We performed ≈200 tests with diagrams containing between 2 and 20 ellipses.

Results. In Fig. 25 we show the number of operations per second achieved in the interactive phase. The two plots refer to the same set of random diagrams but display the results of the number of operations per second versus the number of ellipses (right) and the number of intersection points (left). The number of intersections is important to measure because the number of ellipses essentially places an upper bound on the number of zones in the diagram, whilst the number of intersections captures some information about the complexity of the interaction of the curves. The intention for the library was that it should be capable of obtaining interactive performances suitable for human usage. Therefore, within the plots, we emphasize a limit of 20 operations per second, which represents a reasonable refresh rate for the interactivity; the plot on the right shows that this target rate is easily achieved with diagrams containing up to 10 ellipses.

7. Related work

An important problem that has received much attention is that of the generation problem: given an abstract Euler diagram, decide if there is a concrete diagram which realises the abstract diagram, and if so to produce a drawing of it. The solutions to this problem vary according to the set of well-formedness conditions imposed. In [21], a diagram was called well-formed if it satisfied all of the WF conditions, and a decision procedure and generation algorithm was provided in this case. In other Euler-based generation works [8,26,29,32,34,39], some of these conditions have been relaxed, the basic definitions varied, and such diagrams have been used in the applications described earlier.

In [13], the Euler diagram abstraction problem was solved for well-formed Euler diagrams utilising the single marked point approach in which each zone was tracked via a single marked point. The algorithms applied to weakly reducible diagrams (diagrams obtainable via a sequence of curve additions and removals), and had running time $O(|\mathcal{Z}| + |C| \log |C|)$. In [10] we presented the design and the implementation, in Java, of a library, called EulerVC, which realises these concepts and algorithms for well-formed Euler Diagrams. Furthermore, we utilised this library to develop an application for interactive Euler diagram manipulation 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 example.

The relationship of the Euler diagram abstraction problem with arrangements of Jordan curves in the plane [16,17] was discussed in [10]. We note that in our approach we do not need to store or manipulate graphs, but we work directly with the diagrams using their intersection points and the domain specific data structures, and we obtain a methodology with a straightforward means of implementation.

In [40], an application is presented that interprets an Euler Diagram sketched with a pen or a mouse, and calculates the abstract diagram (this application has been generalised in [35] to Euler Diagrams augmented with graphs). The authors claim complexity that is asymptotically similar to ours, but this claim is not substantiated, with the paper not providing details of how, given two regions r_a and r_b, the system actually computes whether $r_a \cap r_b$ is empty or not. A possible 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. Clark [9] made use of java-area operations to compute the set of zones of a diagram. For each subset of curves, the corresponding area is determined to be present if the area of intersection of the curves is non-empty.

In [42], a methodology is provided which takes as input a set of polygons (i.e. regions determined by sets of non-overlapping curves, which play the role of contours) and outputs a set of non-overlapping polygons (described by the input set), which essentially play the role of minimal regions; 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 [2] (a data representation used to describe polygon models in computer graphics) which captures the set of polygons as a graph (indicating the intersection points) and provides a simple means of traversing the faces of that 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) 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 [3], they prove that the Grünbaum encoding uniquely identifies simple Venn diagrams (i.e. they are well-formed) 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

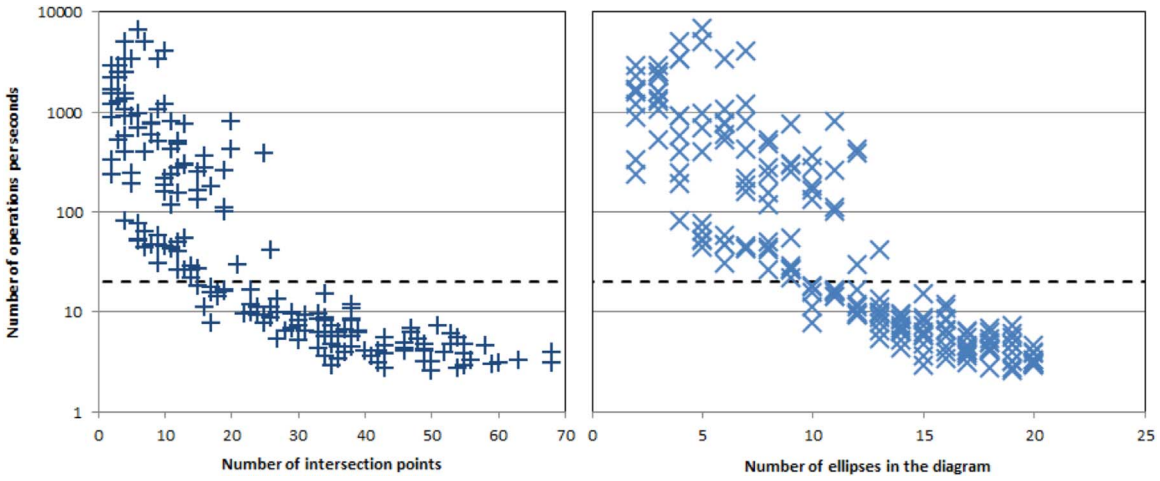

Fig. 25. The performances of the algorithm in a real setting in terms of operation per seconds (so the more operations per second, the better the performance). The horizontal black line represents the interactivity limit of 20 operations per second.

investigating the computation of encodings from the data structures utilised in our algorithms is an interesting line of future investigation. In [27], they extended the work of [4], producing exhaustive lists of simple monotone Venn diagrams that have some symmetry (non-trivial isometry) when drawn on the sphere, and they also prove that the Grünbaum encoding can be used to efficiently identify any monotone Venn diagram. In [1], they developed the foundations for a theory of ED codes (for a particular class of EDs), which could be viewed both as a generalisation/extension of Gauss codes used to encode knot diagrams, as well as a kind of high level abstraction of our basic methodological approach.

8. Conclusion

Euler diagrams, and their generalisations and extensions (e.g. mixed Euler diagram-graph representations), are a common form of representation used for information visualisation when there is an underlying set-relation based requirement. Various Euler diagram generation techniques for automatically creating a drawing of the requirements (usually in the form of an abstract diagram), together with beautification processes to improve the aesthetic quality of the output, have been developed.

The use of Euler diagram based representations is not limited to the presentation of information, however, and they can be used as the basis of the front-end of interactive interfaces. Whilst users may require facilities to create diagrams (e.g. by sketching, by use of pre-defined shapes, by automatic generation), if the users are performing operations that relate to the regions of diagram then identifying those regions quickly (i.e. solving the abstraction problem) is essential in order to enhance response rate. Furthermore, if the diagrams are editable (e.g. if users can modify diagrams by adding, removing or translating curves, say), then the online abstraction problem is the natural problem to address.

Region-based operations have been used in: (i) visual querying – using the set of contours of an Euler diagram to represent query terms and then clicking on a region represents running a query that is the associated Boolean expression over those terms; (ii) visual classification - using the set of contours of an Euler diagram to represent a non-hierarchical classification structure (e.g. providing a view of a file system or tag structure) and then drag and drop of an item (a file, say) into a region provides an immediate classification (placement in file structure or assignment of a set of tags, say); (iii) visual workflows - extending the visual query and classification approaches above to permit the drag and drop of icons representing operations that can be performed on files within the specified classification structure, and the composition of such operations (e.g. to select all items with a set of tags expressed by a region of the diagram, to zip them all into one file and then automatically open an email client with the zipped file as an attachment).

Whilst dealing with well-formed diagrams (whether generating or interpreting) is likely to be preferable from a human perspective (after all, they were introduced with an aim of reducing diagram interpretation errors), there are situations where this is not even achievable (e.g. due to the conditions being so restrictive that certain abstract diagrams cannot be drawn under those conditions), and there may be occasions in which users prefer violation of some of the well-formedness conditions (e.g. in order to be able to draw a diagram with certain symmetries or using fixed geometric shapes, or simply because of aesthetic preference). Therefore, extending methods to permit the relaxation of the drawing conventions (i.e. the well-formedness conditions) is an important task. Furthermore, within an interactive tool setting, if relaxations of the well-formedness conditions are permitted then there may be a need to highlight the areas where these conditions are violated. In particular, if a zone is not connected, being comprised of multiple minimal regions, then the ability to highlight the regions may be required (e.g. in a setting where items are displayed in the regions, then it is desirable to facilitate the visual assessment of the set of all items presented in the regions that comprise the zone).

The previously defined SMPA approach solved the online abstraction problem for well-formed Euler diagrams, but it does not naturally extend to diagrams that permit zones to be disconnected. In this paper, we introduce the MMPA approach which enables this case to be dealt with (but it comes with some additional computational overheads). Furthermore, we provide independent extensions to enable the relaxation of the other well-formedness conditions. We present these as extensions of the MMPA but they could equally well be incorporated into the SMPA approach instead. The complexity analysis presented demonstrates that this approach is no worse than other approaches in the worst case, whilst for generally occurring cases (where the number of regions is less than 2^n for n curves) the approach will fare better, and in particular the online nature of the algorithm can have massive computational savings over an offline approach.

As part of the algorithm extensions discussed, we permit the important capability of dealing with concurrency, including the development of foundational underlying theory for a deeper study of concurrency in this setting. This theory could be used in other settings, such as in the context of Euler graphs (the underlying graph of the diagram) used within automatic diagram generation settings. Furthermore, abstracting the data sets stored as part of the approach would provide an alternative data structure, somewhat reminiscent of a

winged edge data structure used in computer graphics, but which is able to deal effectively with concurrency, for instance. To ensure the ability to represent any abstract diagram, we also present a straightforward method of interpreting generalised Euler diagrams, where multiple curves can have the same label or where curves with self-intersection are permitted.

The algorithms presented in this paper work for general diagrams, for any generic form of curve, but we assume that certain operations (e.g. computing intersection points and types) can be performed quickly (i.e. in constant time) within the complexity analysis. The practical running time may vary according to the forms of curve used, of course. For instance, fixing the geometric shapes to be (arbitrarily rotated) ellipses can lead to faster reaction times than for generic curves since one can take account of the geometric shape when computing the intersections and their types, for instance.

From a practical perspective, the implementation of geometric algorithms is a widely explored field with many interesting problems [23], although most of these problems are bound to the use of limited precision in calculation. 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. It is available as an open source project on GitHub [15]. The library, together with the use of ellipses, permits the manipulation of items placed within the zones of the diagram. It also provides facilities to allow querying of the diagram for the set of zones and the position of the items within the zones.

The library is easily extensible in order to be able to handle different types of contours, provided they can be represented in parametric form, thereby enabling an easy implementation of operations such as choosing a point on a contour, or checking the contour relationships of intersection and containment.

Acknowledgements

We thank all of the anonymous reviewers of different versions of this paper for helping to enhance the quality of the final publication. We acknowledge EPSRC grant EP/J010898/1 Automatic Diagram Generation. No data was produced that could be usefully archived.

References

[1] P. Bottoni, G. Costagliola and A. Fish, Euler diagram encodings, in: Proceedings of Diagrams '12, LNAI 752, 2012, pp. 148–162.
[2] B.G. Baumgart, A polyhedron representation for computer vision, in: Proceedings national computer conference and exposition (AFIPS '75), ACM, New York, NY, USA, 1975, pp. 589–596.
[3] T. Cao, K. Mamakani, F.Ruskey, Symmetric Monotone Venn Diagrams with Seven Curves, in: Proceedings of the Fifth International Conference on Fun with Algorithms, LNCS 6099, 2010, pp. 331–342.
[4] S.-K. Chang, E. Jungert, A spatial/temporal query language for multiple data sources in a heterogeneous information system environment, Int. J. Coop. Inf. Syst. 7 (2) (1998) 167–186. http://dx.doi.org/10.1142/S021884309800009X.
[5] S.-K. Chang, G. Costagliola, E. Jungert, F. Orciuoli, Querying distributed multimedia database data sources in information fusion applications, in: Journal of IEEE transaction on Multimedia, 2004.
[6] S.-K. Chang, W. Dai, S. Hughes, P. Lakkavaram, X. Li, Evolutionary Query Processing, Fusion and Visualization, in: Proceedings of International Conference on Distributed Multimedia Systems, 2002.
[7] S.-K. Chang, Query Morphing for Information Fusion, in: Proceedings of IMAGE: Learning, Understanding, Information Retrieval, Medical, Cagliari, Italy, June 9–10, 2003.
[8] S.C. Chow, Generating and Drawing Area-Proportional Euler and Venn Diagrams, [Ph.D. thesis], University of Victoria, 2007.
[9] R. Clark, Fast Zone Discrimination, in: Proceedings VLL 2007, CEUR, volume 274, 2007, pp. 41–54.
[10] G. Cordasco, R. De Chiara, A. Fish, Interactive Visual Classification with Euler Diagrams, in: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing VL/HCC, IEEE Press, 2009, pp. 185–192.
[11] G. Cordasco, R. De Chiara, A. Fish, V. Scarano, The Online Abstraction Problem for Euler Diagrams, in: Proceedings of Euler diagrams 2012, CEUR 854, 2012, pp. 62–76.
[12] G. Cordasco, R. De Chiara, A. Fish, V. Scarano, FunEuler: an Euler Diagram based Interface Enhanced with Region-based Functionalities, in: Proceedings of Euler diagrams 2012, CEUR 854, 2012, pp. 107–121.
[13] G. Cordasco, R. De Chiara, A. Fish, Efficient on-line algorithms for Euler diagram region computation, Comput. Geom.: Theory Appl. (CGTA) 44 (2011) 52–68.
[14] 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.
[15] R. De Chiara, G. Cordasco, A. Fish, Concrete Euler Diagrams manipulation library GitHub Repository, ⟨https://github.com/rosdec/euler⟩
[16] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, M. Sharir, Arrangements of curves in the plane—topology, combinatorics, and algorithms, in: Theoretical Computer Science, Vol. 92 N. 2, Elsevier Science Publishers Ltd. 1992, pp. 319–336.
[17] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag Inc., New York, 1987.
[18] L. Euler, Lettres a une princesse d'allemagne sur divers sujets de physique et de philosophie, Letters 2 (1775) 102–108 [Berne, Socit Typographique].
[19] A. Fish, J. Flower, J. Howse, The semantics of augmented constraint diagrams, J. Vis. Lang. Comput. 16 (2005) 541–573.
[20] A. Fish, Euler diagram transformations, Graph Transform. Vis. Model. Tech., ECEASST 18 (2009) 1–17.
[21] J. Flower, A. Fish, J. Howse, Euler diagram generation, J. Vis. Lang. Comput. 19 (2008) 675–694.
[22] C.A. Gurr, Effective diagrammatic communication syntactic, semantic and pragmatic issues, J. Vis. Lang. Comput. 10 (1999) 317–342.
[23] C.M. Hoffmann, The problems of accuracy and robustness in geometric computation, Computer 22 (3) (1989) 31–40.
[24] J. Howse, F. Molina, J. Taylor, S. Kent, J. Gil, Spider diagrams a diagrammatic reasoning system, J. Vis. Lang. Comput. 12 (3) (2001) 299–324.
[25] S. Kent, Constraint diagrams: visualizing invariants in object oriented models, in: Proceedings of OOPSLA97, ACM Press, 1997, pp. 327–341.
[26] H. Kestler, A. Muller, T. Gress, M. Buchholz, Generalized Venn diagrams: a new method for visualizing complex genetic set relations, J. Bioinforma. 21 (8) (2005) 1592–1595.
[27] K. Mamakani, W. Myrvold, F. Ruskey, Generating simple convex Venn diagrams, J. Discret. Algorithms 16 (8) (2012) 270–286.
[28] N. Henry Riche, T. Dwyer, Untangling Euler diagrams, IEEE Trans. Vis. Comput. Graph. 16 (6) (2010) 1090–1099.
[29] P. Rodgers, L. Zhang, A.Fish, General Euler Diagram Generation, in: Proceedings of the 5th International Conference on Diagrams 2008, Vol. 5223 of LNAI, Springer-Verlag, 2008, pp. 13–27.
[30] F. Ruskey, A survey of Venn diagrams, Electronic Journal of Combinatorics. ⟨www.combinatorics.org/Surveys/ds5/VennEJC.html⟩, 1997.
[31] A.Shimojima, Inferential and expressive capacities of graphical representations: survey and some generalizations, in: Proceedings of the 3rd International Conference on the Theory and Application of Diagrams, Vol. 2980 of LNAI, Springer-Verlag, 2004, pp. 18–21.
[32] P. Simonetto, D. Auber, D. Archambault, Fully automatic visualisation of overlapping sets, Comput. Graph. Forum (EuroVis09) 28 (June (3)) (2009) 967–974.
[33] G. Stapleton, J. Masthoff, J. Flower, A. Fish, J. Southern, Automated theorem proving in Euler diagrams systems, J. Autom. Reason. 39 (2007) 431–470.
[34] G. Stapleton, P. Rodgers, J. Howse, L. Zhang, Inductively generating Euler diagrams, IEEE Trans. Vis. Comput. Graph. 17 (1) (2011) 88–100.
[35] G. Stapleton, B. Plimmer, A. Delaney, P. Rodgers, Combining sketching and traditional diagram editing tools, ACM Trans. Intell. Syst. Technol. 6 (1) (2015) 10.
[36] N. Swoboda, G. Allwein, Using DAG transformations to verify Euler/Venn homogeneous and euler/venn fol heterogeneous rules of inference, J. Softw. Syst. Model. 3 (2) (2004) 136–149.
[37] J. Thièvre, M. Viaud, A. Verroust-Blondet, Using Euler Diagrams in Traditional Library Environments, in: Euler Diagrams 2004, Vol. 134 of ENTCS, 2005, pp. 189–202.
[38] J. Venn, On the Diagrammatic and Mechanical Representation of Propositions and Reasonings, Phil. Mag, 1880.
[39] A. Verroust, M.-L.Viaud, Ensuring the Drawability of Extended Euler Diagrams for up to eight Sets, in: Proceedings of the 3rd International Conference on the Theory and Application of Diagrams, volume 2980 of LNAI, Cambridge, UK, Springer, 2004, pp. 128–141.
[40] M. Wang, B. Plimmer, P. Schmieder, G. Stapleton, P. Rodgers, 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, IEEE Press, 2011, pp. 75–82.
[41] Li. Xin, S.K.Chang, An interactive visual query interface on spatial/temporal data, in: Proceedings of the Tenth International Conference on Distributed Multimedia Systems, 2004.
[42] K. Weiler, Polygon comparison using a graph representation, Computer Graphics (SIGGRAPH '80 Proceedings), 14(3):10–18, July 1980.
[43] R.J. Wilson, Introduction to Graph Theory, 4th edition, Longman, 1996.