Interactive Visual Classification with Euler Diagrams

Gennaro Cordasco, Rosario De Chiara
ISISLab - Dipartimento di Informatica
ed Applicazioni "R.M. Capocelli"
Università di Salerno - ITALY
{cordasco, dechiara}@dia.unisa.it

Andrew Fish*
School of Computing, Mathematical
and Information Sciences,
University of Brighton - UK
Andrew.Fish@brighton.ac.uk

Abstract. *We present the theoretical foundation, the design and the implementation of a library, called EulerVC to interactively handle Euler diagrams for the purposes of resource management. Fast on-line algorithms to interpret wellformed diagrams have been developed utilising a new notion of marked points to keep track of the zone sets. The interface allows the construction of overlapping ellipses to represent categories together with the drag and drop of resources in order to categorise them. A visual indicator can be used to show if the diagram under construction is not wellformed to assist in reducing user mistakes, and sets of tags can be assigned to resources upon export. The generic approach is demonstrated via an integration with the bookmarking site* del.icio.us.

1 Introduction

Hierarchical visualization is a perfectly natural method for organizing and presenting information since in real life objects can be just in one place. However, given that this metaphor breaks down in the virtual world, that categorizing resources is a difficult problem where users may wish to categorize resources differently according to factors such as their current tasks, different methodologies have developed. A common approach to enable a more general categorization system is that of free form tagging, where users can tag resources arbitrarily and then mechanisms for searching for resources in the flat tag space need to be developed, as well as visualization techniques such as tag clouds.

We take the approach of preferring a non-hierarchical representation that both facilitates the storage and categorization of resources and is utilized as the visualization of existing resources or the results of a user query. VennFS [6] is a tool that facilitates the construction of Venn diagrams for categorization purposes. However, for Venn diagrams with several contours the diagrams can become cluttered, and navigation problems and computational difficulties can occur due the large numbers of regions present (which is exponential in the number of contours or categories). In [2] a novel tree-based interface, which was essentially a reification of Euler diagrams, was developed for the purposes of non-hierarchical categorization; the idea was that the common tree based interface would not be too challenging to learn and that it could utilize the power of the Euler diagram model without the inherent diagrammatic problems that can arise when many categories are involved. This interface has been integrated with *del.icio.us* to give Eul.icio.us [3] and *Flickr* to give Eulr [4] to assist users in categorising their bookmarks and photos and facilitating the export of the associated tags to the respective websites.

In this paper we present the theoretical foundation, the design and the implementation of a library, called EulerVC to interactively handle Euler diagrams for the purposes of resource management. We have developed fast on-line algorithms which enable the system to efficiently keep track of the zones present in the diagram, thereby overcoming the aforementioned computational difficulties; in [5] a slow, brute force algorithm was presented. Due to the efficiency of the algorithms developed, the library does not limit the number of overlapping categories permitted and can easily deal with hundreds of them. Of course the effects of visual clutter [10] will reduce the comprehensibility of such large diagrams, and if one really wished to deal with diagrams of this size in practice then navigation techniques would need to be developed or incorporated. In Section 2 we recall necessary background on Euler diagrams and indicate the effects of including tags. Then, in Section 3 we provide definitions related to viewing diagram construction and manipulation as a sequence of contour additions and removals, develop some necessary theory and then we provide examples in Section 3.1.1 together with an outline of the main algorithms in Section 3.1.2. The complete detailed formal versions of the algorithms cannot be provided for space reasons, but the examples and outline should provide a clear exposition of the main ideas involved. Furthermore, we have implemented a general purpose Java library called EulerVC that can be used to interactively draw Euler diagrams by adding or removing ellipses. As proof of

*Funded by UK EPSRC grant EP/E011160: Visualisation with Euler Diagrams.

concept we have exploited this library to implement an application to facilitate the categorization of bookmarks that a user keeps on *del.icio.us*. In Section 4 we present this prototype which allows the user to quickly construct Euler diagrams using ellipses, and has drag and drop facilities to enable the categorization of resources, together with import and export facilities to *del.icio.us*. The goal of developing a library is to make the algorithmic functionality easily available to applications that can be developed. Not only can the library be extended to incorporate other algorithmic functionalities, but a suite of applications for the purpose of user-categorization based on Euler diagrams and related representations can now be developed.

2 Background

We recall Euler diagram definitions; detailed formal definitions can be found in [7] for instance. A *concrete Euler diagram* is a pair $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ where $\mathcal{C} = \mathcal{C}(d)$ is a set of labeled *contours* (closed curves) in the plane and $\mathcal{Z} = \mathcal{Z}(d)$ is the collection of *concrete zones* z determined by being inside a set of contours $X_z \subseteq \mathcal{C}(d)$ (X_z is called a *zone descriptor*) and outside the rest of the contours of the diagram. That is,

$$z = \bigcap_{c \in X_z} interior\,(c) \cap \bigcap_{c \in \mathcal{C}(d) - X_z} exterior\,(c)\,,$$

for each $X_z \subseteq \mathcal{C}(d)$, provided this region is non-empty. Each concrete diagram contains a zone o, called the *outer zone*, which is exterior to all the contours (that is, $X_o = \emptyset$).

The following set of topological or geometric conditions, called *wellformedness conditions*, are often imposed on concrete Euler diagrams in order to try to prevent user misunderstanding: the contours are simple closed curves; each contour has a single, unique label; contours meet transversely (so no tangential meetings or concurrency); at most two contours meet at a single point; each concrete zone is a *minimal region* (a minimal region is a connected component of the complement of the contour set, so this condition means that zones cannot be disconnected). A concrete Euler diagram is *wellformed* if it satisfies all of these wellformedness conditions.

An *abstract Euler diagram* d is a pair $\langle L, \mathcal{Z} \rangle$, where L is a set of labels and $\mathcal{Z} = \mathcal{Z}(d)$ is a subset of $\mathcal{P}(L)$, the power set of L, called the *abstract zone set* of d. The *abstraction* of a concrete Euler diagram d is the abstract Euler diagram whose label set L is the set of labels of $\mathcal{C}(d)$ and whose abstract zone set consists of exactly the sets of containing contours for the concrete zones of d. Note that we are abusing notation, using $\mathcal{Z}$ for both abstract and concrete zones. In the context of wellformed Euler diagrams, there is a natural correspondence between these two concepts. In order to aid the exposition in this paper we choose to blur the distinction: in our algorithms we are generally dealing with concrete Euler diagrams, but in fact we keep track of the concrete contours together with the abstract zone set (which can be used to identify the concrete zone set) rather than the concrete zone set as stated.

We wish to view the zones of a concrete Euler diagram as a repository in which to place resources, such as `urls` and so we extend the basic notion of Euler diagrams to capture the placement of items in the diagram; this is similar to a unitary alpha Spider diagram [8].

Definition 1 *Let $\mathcal{L}$ be a set of item labels. Then a* concrete Euler diagram with items *in $\mathcal{L}$ is a concrete Euler diagram $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ together with a set of points p_i in $\overline{\mathcal{C}}$, the complement of (the images of) the contours in the plane, each of which is labelled by a distinct $\ell_i \in \mathcal{L}$. An* abstract Euler diagram with items *in $\mathcal{L}$ is an abstract Euler diagram $\langle L, \mathcal{Z} \rangle$ together with a labelling function $f : \mathcal{Z} \to \mathcal{P}(\mathcal{L})$. The* abstraction *of a concrete Euler diagram with items is the abstract Euler diagram with items which is the abstraction of the concrete Euler diagram d' such that for each labelled point p, if p is in concrete zone z' of d' then the label of p is associated with the abstract zone for z'.*

We can easily associate a set of tags to the items (e.g. resources) of an Euler diagram with items; this is necessary for the purposes of exporting to tagging systems.

Definition 2 *Let d' be a concrete Euler diagram with items in $\mathcal{L}$ that has a zone z' containing a point p labelled by $r \in \mathcal{L}$. Suppose that d is the abstraction of d' with label set L, and that $z = \{l_1, \ldots, l_n\}$, with each $l_i \in L$, is the abstract zone corresponding to z'. Then the* set of tags *associated with any of r, p, z', or z is $\{l_1, \ldots, l_n\}$.*

Thus the association of a set of tags to a resource placed in a zone of a concrete Euler diagram amounts to computing the corresponding abstract zone, and this computation needs to be efficient in order for any application utilising it to be of practical use. We have developed such algorithms for wellformed Euler diagrams and we show a worked example in the next section.

3 Fast diagram interpretation

We present fast algorithms to solve the following problems, given a wellformed concrete Euler diagram d:

1. evaluate whether or not the process of adding a new contour or removing an existing contour yields a wellformed diagram.
2. compute the abstract Euler diagram for d [1].

First we introduce terminology related to the processes of adding, or removing, contours to, or from, a diagram.

Definition 3 *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed concrete Euler diagram. Then*

- *If $A \notin \mathcal{C}$ then the effect of the* addition of contour A *to d is denoted $d + A$. Each zone of d that is completely covered by A by this operation is called a* covered zone

[1] i.e. the abstract information of which zones are present in the diagram, or equivalently a set of contours describing the zone.

(i.e. if $z \in \mathcal{Z}(d)$ then $z \subset interior(A)$ in $d + A$). Each zone of d that is partially covered by A is called a split zone *(i.e. if $z \in \mathcal{Z}(d)$ then z intersects both $interior(A)$ and $exterior(A)$ non trivially in $d + A$) and we say that z is split by A generating two zones, $z' = z \cap exterior(A)$ and $z'' = z \cap interior(A)$ in $d + A$. Since the zone z' has the same zone descriptor as z, we say that A generates one new zone, corresponding to z'', in $d + A$.*

- *If $B \in \mathcal{C}$ then the effect of the* removal of contour B *is denoted $d - B$. In this case, each zone of $d - B$ that is completely covered by B upon its addition (yielding d) is called a* covered zone*; each zone of $d - B$ that is partially covered by the addition of B is called a* split zone.

Remark 1 *These concepts such as covered and split zone are relative to the contour under consideration. Furthermore, when considering contour removal, viewing B as if it was the last contour added, the zones z' and z'' that are split by B need to be merged in order to obtain $d - B$.*

We wish to consider applications where Euler diagrams are constructed using these natural operations of adding and removing contours, and to preserve the wellformedness of the diagrams upon such transformations, so it makes sense to consider the class of Euler diagrams that are constructible via these operations. *Reducible* Venn diagrams are defined in [11] and we first provide the natural extrapolation of that definition to reducible Euler diagrams.

Definition 4 *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed Euler diagram where $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$. Then d is said to be* reducible *iff there exists a permutation $\pi \in S_n$ such that all of the following n diagrams are wellformed:*
$d_1 = \langle \{C_{\pi_1}\}, \mathcal{Z}_1 \rangle$,
$d_2 = \langle \{C_{\pi_1}, C_{\pi_2}\}, \mathcal{Z}_2 \rangle$,
$\ldots$,
$d_n = \langle \{C_{\pi_1}, C_{\pi_2}, \ldots, C_{\pi_n}\}, \mathcal{Z}_n \rangle = d = \langle \mathcal{C}, \mathcal{Z} \rangle$,
where π_i denotes $\pi(i) \in \{1, \ldots, n\}$.

However, there exist Euler diagrams which are not reducible (i.e. cannot be constructed as a sequence of contour additions keeping a wellformed diagram at each step) but can be constructed via a sequence of additions and removals of contours, always keeping a wellformed diagram at each step. For instance, the example of the symmetric Venn(5) which is drawn with ellipses is irreducible [11] but it can be constructed via a sequence of additions and removals of contours. We will call this broader class of diagrams *weakly reducible* Euler diagrams.

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

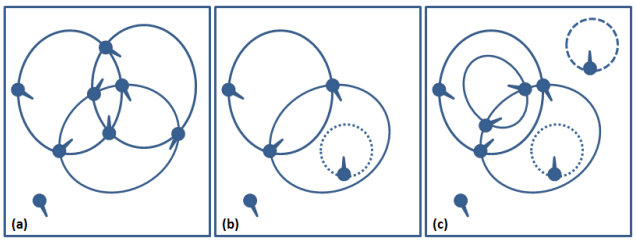

Figure 1: Euler diagrams with different numbers of connected components, which are emphasized using different strokes: (a) 1 connected component, 6 intersection points and 8 zones; (b) 2 connected components, 2 intersection points and 5 zones; (c) 3 connected components, 4 intersection points and 8 zones.

We first define properties of weakly reducible Euler diagrams that we can easily compute and relate then to the property of wellformedness.

Definition 6 *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a Euler diagram. Then an* intersection point *of d is a point of intersection of the contours in $\mathcal{C}$. The number of intersection points of d is denoted by $ip(d)$. The set of contours in d can be partitioned into the connected components of d (i.e. the maximal sets of intersecting contours). We denote the number of components of d by $cc(d)$.*

Theorem 1 *Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a weakly reducible Euler diagram, then $|\mathcal{Z}| = ip(d) + cc(d) + 1$.*
The proof is omitted due to space constraints.

Figure 1 shows three examples of Euler diagrams with different numbers of connected components. Theorem 1 will ensure that we can choose a set of *marked points* for a wellformed Euler diagram, one for each zone, and be able to update this set of marked points coherently as we alter the diagram via the addition or removal of contours. Note that the little pointers in the figures throughout the paper are simply visual indicators for the reader showing which zone is associated with which marked point. That is, the marked points are actually the points shown on the contours, whilst their pointers indicate visually which zone those points are marking.

3.1 Interpreting Euler Diagrams with ellipses

Although our interpretation algorithms are applicable to any wellformed concrete Euler diagrams, we specialize to the case where the contours are arbitrarily oriented ellipses. This enables an even faster construction and interpretation method due to the geometric constraints imposed. Henceforth, when we refer to "diagrams" we refer to "concrete Euler diagrams, all of whose contours are ellipses".

Now, given two ellipses A and B, one can quickly find:

- their intersection points (in particular, since in a wellformed diagram tangential points are not allowed, we will have $0,\ 2$ or 4 intersection points);

- their relationship (that is, if they overlap, if one is contained [2] in the other or if they are disjoint);

We present here the main ideas of our algorithm for zone computation. Let d be a wellformed Euler diagram, and suppose we wish to add an ellipse A to d (where $A \notin \mathcal{C}(d)$) or remove an ellipse A from d (where $A \in \mathcal{C}(d)$), then:

1. We can represent each zone of d using a single marked point; that is, we associate to each zone $z \in \mathcal{Z}(d)$ a single point $m(z) \in \mathbb{R}^2$ such that $m(z)$ belongs to $cl(z)$ (where $cl(z)$ denotes the closure in $\mathbb{R}^2$ of the open region z).
2. By analyzing the intersection points generated by A (that is, the intersection points between A and each ellipse in $\mathcal{C}(d) - A$) we can identify the zones split by A (cf. Definition 3). Then for each split zone we generate a new zone (in the case of addition) or merge two corresponding zones (in the case of removal).
3. We can quickly update the marked point associations upon the addition/removal of A.
4. Using the marked points, we can quickly check whether a zone, which is not split by A, is covered by A (in time constant with respect to the number of ellipses present in the the diagram) and accordingly update its zone descriptor.

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

3.1.1 Some Illustrative Examples

Before providing the description of our solution, we describe it informally, using some examples. The first example, depicted in Figure 2, shows a diagram generated as a sequence of four ellipse additions, starting from an empty diagram. In the following, we will talk about the diagrams d_1, d_2, d_3 and d_4 for the diagrams with $1, 2, 3$ and 4 ellipses in, depicted in the Figure 2. When the first ellipse A is added (cf. Figure 2 (a)) there are no intersection points. In this case the new ellipse generates exactly one new zone (having zone descriptor $\{\mathrm{A}\}$), which is obtained by splitting the *outer zone*. Specifically, d_1 has 1 ellipse A and 2 zones described respectively by $\emptyset$ and $\{A\}$. The new zone's marked point is chosen as an arbitrary point on the ellipse

[2] More technically, we mean if the region bounded by one contour is contained in the interior of the region bounded by the other contour.

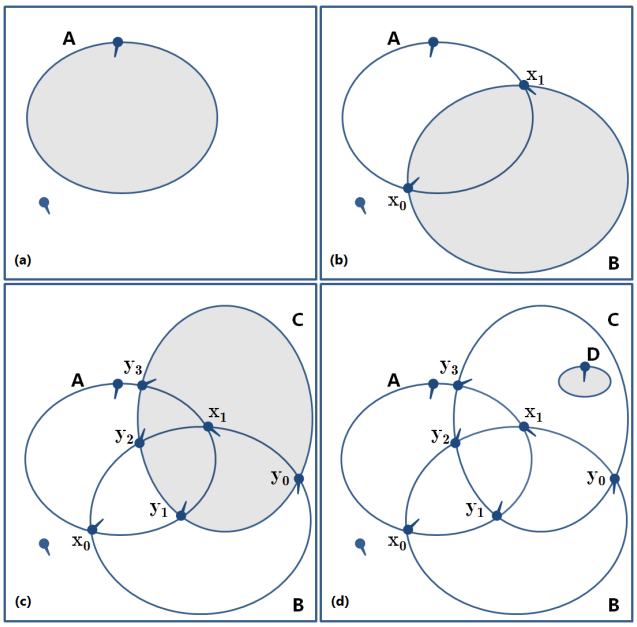

Figure 2: A sequence of ellipse additions with marked points associations.

A. The second ellipse B intersects the diagram d_1 in two points, x_0 and x_1 and generates two new zones (cf. Figure 2 (b)). The new zones are obtained by splitting both of the zones described by $\{A\}$ and $\emptyset$ (the *outer zone*) in d_1. Specifically, d_2 has 2 ellipses A and B and 4 zones described by $\emptyset, \{A\}, \{B\}$ and $\{A, B\}$. The marked points associated with the new zones are computed using a simple method which allows us to maintain the property that each marked point belongs to the closure of the described zone. The third ellipse C intersects the diagram d_2 in four points, y_0, y_1, y_2 and y_3, and generates four new zones (cf. Figure 2 (c)). The split zones are $\emptyset, \{A\}, \{A, B\}$ and $\{B\}$.

In this case the previous marked point assignments need to be changed: the points x_1 and y_0 are swapped. That is, the point x_1, previously assigned to the zone $\{B\}$ in d_2, is now assigned to the zone $\{B, C\}$ in d_3 and accordingly the intersection point y_0, between C and d_2, is assigned to the zone $\{B\}$ previously marked with x_1. Finally, Figure 2 (d) shows what happens when a new contour D, which does not intersect any other ellipse in d_3, is added, giving d_4. The ellipse D belongs to a new connected component, splits the zone $\{C\}$ in d_3 and generates a new zone $\{C, D\}$. The new zone's marked point is chosen as an arbitrary point on the ellipse D.

The second example, depicted in Figure 3, shows two significant cases. The starting diagram is d_4 (cf. Figure 2 (d)). The first case depicted in Figure 3 (a) shows the addition of a new ellipse E to d_4 to give d_5. This example explains why, during an addition/removal operation, we not only need to identify the set Z_s of split zones and the corresponding set of new zones (shown in light grey in the figure), but we also need to identify the set Z_c of zones (shown

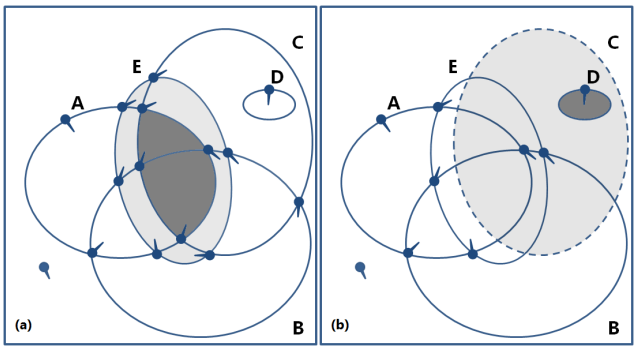

Figure 3: (a) The curve E is added to d_4 to give d_5; (c) The ellipse C is removed from d_5 to give d_4'.

in dark grey in the figure) which are covered by E. The zones in Z_c, which need their zone descriptor sets updated, can be quickly identified by checking if their marked points lie in the interior of E. The second case, depicted in Figure 3 (b) shows the removal of the ellipse C from d_5 to give d_4' indicating why identifying the set of split and covered zones is enough to update the diagram appropriately. In particular in order to obtain $d_4' = d_5 - C$ each split zone z' having zone descriptor $X_{z'} \in \{\{C\}, \{B, C\}, \{B, C, E\}, \{A, B, C, E\}, \{A, C, E\}, \{C, E\}\}$ needs to be merged with the corresponding zone z'' such that $X_{z''} = X_{z'} - C$. Moreover, the zone with descriptor $\{C, D\}$ in d corresponds to the zone that is covered by C, with descriptor $\{D\}$, in $d - C$ and so the zone descriptor is updated to become $\{D\}$.

3.1.2 Algorithm description

For space reasons we describe only the algorithm for ellipse addition, omitting the detailed description of the algorithm for ellipse removal.

Let $d = \langle \mathcal{C}, \mathcal{Z} \rangle$ be a wellformed Euler diagram and A an ellipse such that $A \notin \mathcal{C}$. Then the high-level of the algorithm **NewEllipse**(d,A) is depicted in Figure 4: reading clockwise from the top, the gray arrows indicate the main sequence of steps in the algorithm flow, the black outlined arrows indicate the relationships between the steps and the data structures shown in the middle of the diagram, and finally the dashed boxes show an example of the diagram being handled in that step.

Compute A's relationship with d First of all the algorithm computes the collection $Over(A)$ of all the ellipses in $\mathcal{C}$ that properly overlap with A (that is $Over(A) = \{C \in \mathcal{C}$ such that $A \cap C \neq \emptyset\}$), the collection $Cont(A)$ of all the ellipses in $\mathcal{C}$ that properly contain A (that is $Cont(A) = \{C \in \mathcal{C}$ such that $C \notin Over(A)$ and $int(C) \cap int(A) = int(A)\}$, where $int(A)$ denotes the interior of A) and the set $Inter(A)$ of intersection points between A and the ellipses in $Over(A)$.

Check if $d + A$ is wellformed The algorithm applies Theorem 1 in order to assess if the diagram $d + A$ is wellformed. If $d + A$ is not wellformed A is rejected. This check is also performed at drawing time in order to give visual feedback about the wellformedness of the current diagram to the user. The examples at the right of Figure 4 indicate the three types of failure of wellformedness that could occur using ellipses.

Compute Split Zones There are two cases to consider: if $Over(A) = \emptyset$ then no intersections are created by the addition of A, and this means that A belongs to a new connected component. Thus, A splits only the zone described by contours in $Cont(A)$ (see for instance contour D in Figure 2 (d)). On the other hand, if $Over(A)$ is not empty, then A splits several zones (see for instance contour B in Figure 2 (b)). The algorithm computes the split zones by analyzing all of the intersection points generated by A (i.e., the points in $Inter(A)$). The example at the bottom right of Figure 4 shows the split zones for A in grey.

Generate new Zones and Update Marked Points The algorithm first generates the new zones by adding $\{A\}$ to each split zone descriptor and then computes the marked points for the split zones and the corresponding new zones. In particular, if the marked point associated with the split zone belongs to A then a *swapping* operation is performed (cf. Figure 2 (c)). The example at the bottom left of Figure 4 shows the new zones for A in dark grey. A larger version of the "update marked points" example shown in Figure 4 can be found in Figure 3 (a). Notice that if A belongs to a new connected component (that is, if $Over(A) = \emptyset$), then only a new zone is generated. This zone corresponds to the interior of A and its marked point is chosen as an arbitrary point on the ellipse A (cf. Figure 2 (a) and (d)). Notice that the association of marked points to zones is non deterministic, and different orderings of curve addition/deletion can give rise to different marked point associations.

Update Covered Zones Eventually the algorithm updates the zone descriptor of each zone covered by A. In more detail, for each non-split zone z the algorithm checks if z belongs to A by checking where z's marking point lies. Since z is not split by A, checking one single point is enough to evaluate the relationship between z and A. The example at the top left of Figure 4 shows the zones covered by A in grey.

The algorithm for ellipse removal is very similar to that for ellipse addition, but we briefly indicate their differences. During the first step, the algorithm for ellipse removal computes the relationship between $d - B$ and the contour B to be removed. Then the algorithm applies Theorem 1 in order to check if $d - B$ is wellformed. The algorithm computes the zones of $d - B$ that are split by B, effectively viewing B as if it was the last contour added to d. Then, each split zone z, having descriptor X_z, is merged with its corresponding new zone z' described by $X_z \cup B$. Each merged zone has its marked point obtained by arbitrarily selecting one of the two marked points of the two zones that were merged. The last step is to remove B (whenever it is present) from each zone descriptor of non-split zones.

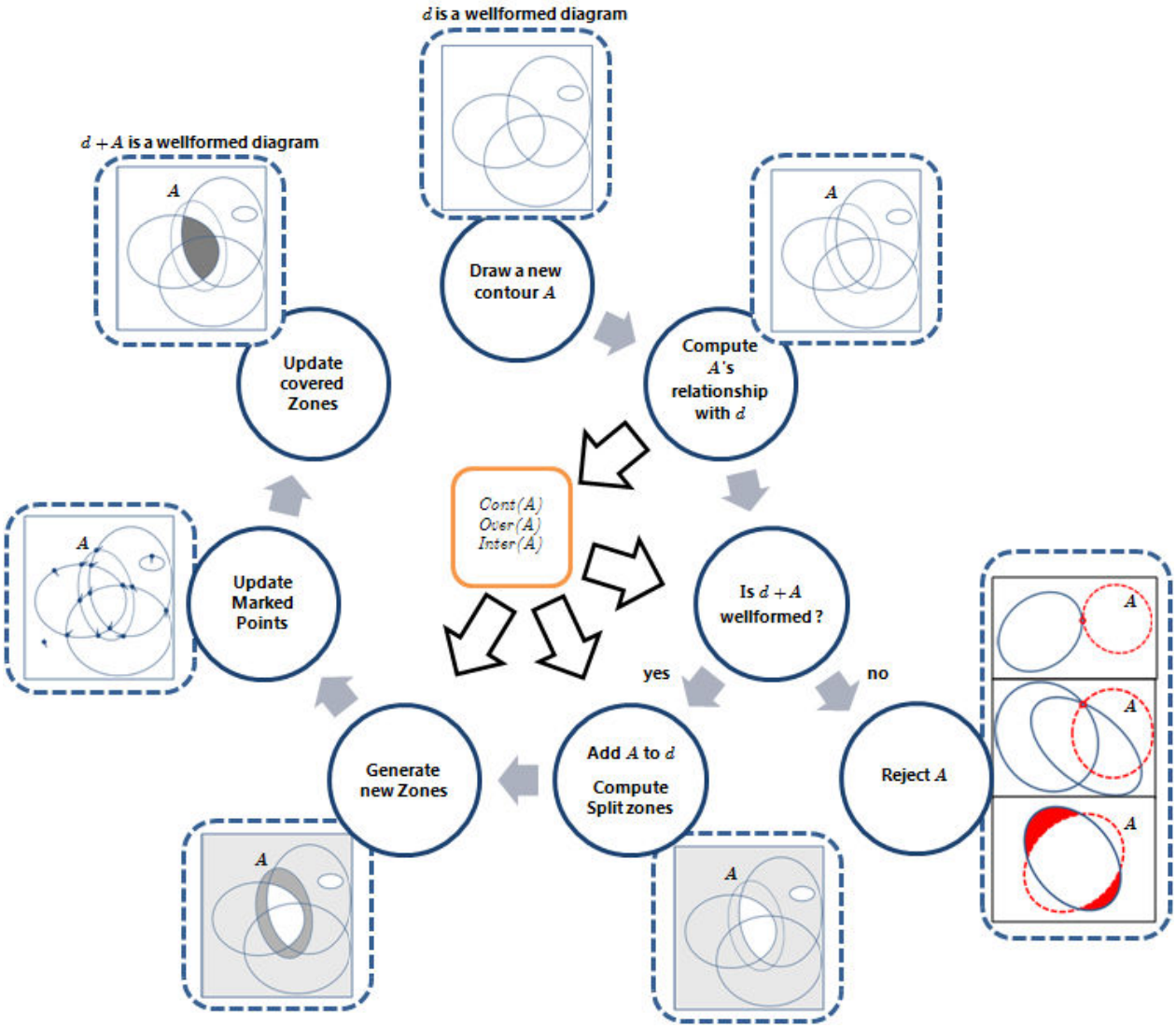

Figure 4: Flow of the algorithm **NewEllipse**(d, A).

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

i) *If $A \notin \mathcal{C}$ and $d' = d + A$ is wellformed, then the procedure **NewEllipse**(d, A) computes the new set of zones $\mathcal{Z}'$ of $d' = \langle \mathcal{C} \cup A, \mathcal{Z}' \rangle$.*

ii) *If $B \in \mathcal{C}$ and $d' = d - B$ is wellformed, then the procedure **DeleteEllipse**(d, B) computes the new set of zones $\mathcal{Z}'$ of $d' = \langle \mathcal{C} - B, \mathcal{Z}' \rangle$.*

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

The proof is omitted due to space constraints.

In order to understand the speed-up obtained using our algorithms, we intuitively compare with a simple naive approach which, each time a new contour A is added to a given diagram $d = \langle \mathcal{C}, \mathcal{Z} \rangle$, checks whether each zone is split by A. In this naive approach the existence of a zone is achieved by a function **CheckZone** which depends on the properties of the contours that are present in the diagram. For example, if all of the contours are ellipses then we need to solve a specific system of $|\mathcal{C}|$ inequalities in order to compute the presence of each zone. The function **CheckZone** requires a significant amount of time, clearly non constant with respect the number of contours present in d. Let $f(|\mathcal{C}|)$ denote the time taken by the function **CheckZone**. Then the complexity of the naive algorithm is $O(f(|\mathcal{C}|) \times |\mathcal{Z}|)$ which is worse than $O(|\mathcal{C}| log |\mathcal{C}| + |\mathcal{Z}|)$.

4 A test implementation

The theory illustrated in the previous sections enables the construction of an efficient implementation which allows the user to utilise Euler diagrams as a method of information management. For this purpose we have implemented the algorithms as a general purpose Java library

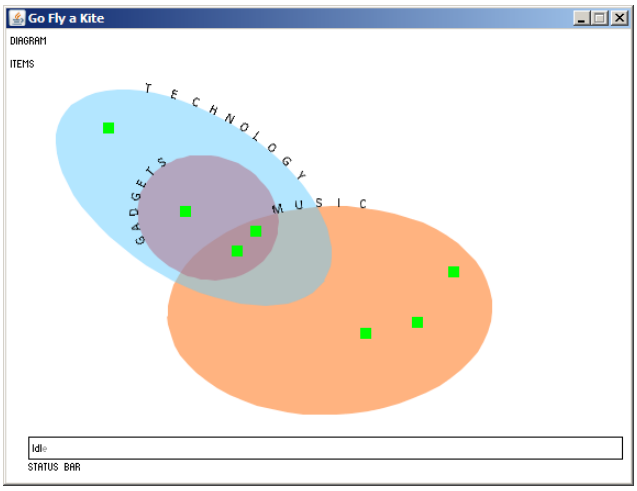

Figure 5: A snapshot of the application prototype: three ellipses are represented together with items.

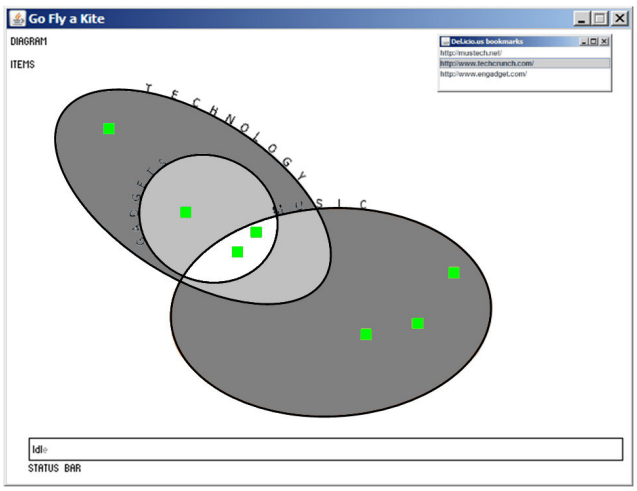

Figure 6: Importing user's bookmarks from delicious.

called EulerVC. An Euler diagram consisting of arbitrary ellipses can be constructed and it is possible to record the placement of items with the regions determined by the ellipses in order to categorize them. Each ellipse can be characterized by a few geometric parameters: a center coordinate, two axes and an angle of rotation; every item has a set of coordinates (but additional parameters depending on the scenario may be added).

The library allows addition, update and removal of ellipses to or from an Euler diagram and coherently updates a flag stating whether the diagram is wellformed or not. This flag can be used to provide the user with visual feedback when they are in the process of adding, updating or removing an ellipse in order to clearly depict if the operation is permitted (in the sense of constructing a wellformed diagram).

4.1 A scenario: *del.icio.us*

As proof of the generality of the library we implemented a small application that allows the user to easily tag the bookmarks he/she keeps on *del.icio.us*. Briefly, *del.icio.us* [1] is a social bookmarking website that allows the user to store bookmarks and to categorise them by using free form tagging. The purpose of our application is to let the user drag bookmarks from *del.icio.us* and drop them onto the diagram: placing a bookmark on the diagram naturally associates to it a set of tags. Figure 5 shows the application's user interface depicting an Euler diagram and some bookmarks. It is worth noting that both the ellipses in the Euler diagram and the bookmarks are drawn and manipulated interactively, exploiting the functionalities offered by the library.

Ellipses To draw a new ellipse the user just has to click and drag on the diagram to define one of the axes of the ellipse whilst using the mouse wheel it is also possible to alter the other axis; the mouse position also defines the rotation angle of the ellipse. The application provides the user with feedback indicating if the diagram constructed by adding the new ellipse is wellformed or not, by colouring it blue or red, respectively. In the current version, the user is prevented from adding an ellipse if the resulting diagram would be not wellformed (e.g. it induces disconnected zones). Every ellipse already present in the diagram can be interactively modified by modifying its placement, by a rotation, or by removing it; in every case wellformedness is enforced by not permitting operations that violate it. It is possible to change ellipses' colour and name (which corresponds to the tag associated with the items that are placed in the interior of the contour).

Bookmarks Bookmarks can be easily added to the diagram by using the drag and drop functionality. Exporting to *del.icio.us* is straightforward: every bookmark placed in the diagram corresponds to an item of a *concrete Euler diagram with items* (cf. Definition 1) and accordingly it is associated with the set of tags given by the zone in which the item lies (cf. Definition 2). The application offers the possibility of importing bookmarks from *del.icio.us*: when the user selects a bookmark, the diagram is redrawn, displaying with a brighter color the zone that corresponds to the set of tags associated with it. This indicates to the user where the bookmark would currently be placed. In Figure 6 the selected bookmark has three associated tags and the zone corresponding to the three tags is the brightest one.

5 Conclusion

Resource organisation is an everyday task for the majority of people. We have developed a building block, called EulerVC, to develop a new class of interfaces to assist users in such tasks, enabling the construction of Euler diagrams, using ellipses, and facilitating multiple tagging of resources via the simple drag and drop process. It uses the placement of resources in zones to determine the tag sets, associating contour labels with tags and zones with the set of tags in their containing contour set. For demonstration purposes,

we have integrated the interface with *del.icio.us* enabling both the import and export of bookmarks and their tag sets. Efficient algorithms have been developed to interpret diagrams in an on-line manner (that is to compute the zone sets or their change under the the addition or removal of contours).

Now, the appeal of diagrammatic representations can often be questioned when one considers their use on a large scale. However, provided there are not too many categories with large amounts of overlap (i.e. too many complex clusters) then users should be able to focus on the smaller clusters and the idea is that comprehension of each of the smaller clusters should then be manageable; the development of navigation aids should improve the situation further. As an alternative, the integration with the EulerView representation [2] will allow the use of Euler diagrams, as described here, for the small scale where they are likely to provide the most benefits, together with the alternative EulerView representation to enable interaction with larger scale structures.

The wellformedness checking facilities of our application also gives instant user feedback on potential problems of comprehension that the diagrams constructed might cause; this may be especially useful in the future within the context of user's sharing their diagrams with other people. The restriction to wellformed diagrams is not unreasonable for diagrams constructed with ellipses that use a small numbers of contours (e.g. there is a wellformed symmetric Venn diagram on 5 contours [11] that can be constructed), or which only have clusters with such small numbers of contours. This choice of wellformedness conditions imposed is the set imposed in [7] where the conditions under which an abstract diagram is drawable are presented, and an algorithm to automatically generate wellformed Euler diagrams presented; this will be beneficial for future developments since the integration of these facilities will enable automatic diagram construction from tagged resources. Now, a user might want to use non-wellformed diagrams to express relationships more compactly, for instance, or because diagram alterations (e.g. adding or removing a contour, or altering the relationships between contours) causes the wellformedness conditions to fail. For this reason, the algorithms are currently being extended to encompass non wellformed cases, although this extension is not straightforward.

The implementation of the algorithms as a general purpose Java library will assist in future plans of the development of a new visual interface to enable users to categorise all of their resources and to export the tag sets to different sites such as *del.icio.us* for bookmarks, *Flickr* for photos, or their own file system for general resources. The EulerVC library is just the base on which to build different applications to further investigate the use of Euler diagrams in different contexts. There are many future avenues for research such as investigating the *cooperative tagging of resources* [9] and the *interactive exploration of Euler diagrams*. Cooperative tagging is commonly performed on social networking websites where a wide range of resources (e.g. photos, bookmarks, movies, songs etc. . .) can be tagged not only by the owner but also by the whole community, building a picture of the multiplicity of meanings a single piece of information can have. An application for the interactive exploration of Euler diagrams could help researchers to investigate the effects of contours addition and removal (by indicating when diagrams will become not wellformed), assisting in theoretical understanding of concepts such as reducibility and weak reducibility.

Acknowledgements. The authors would like to thank the anonymous referees for their very helpful comments which enabled a significant improvement in the presentation of the work.

References

[1] del.icio.us. http://del.icio.us.

[2] Rosario De Chiara and Andrew Fish. Eulerview: a non-hierarchical visualization component. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*. IEEE Computer Society, 2007.

[3] Rosario De Chiara and Andrew Fish. eul.icio.us: Euler diagrams for del.icio.us. In *12th International Conference Information Visualisation*. IEEE Computer Society, 2008.

[4] Rosario De Chiara, Andrew Fish, and Salvatore Ruocco. Eulr: a novel resource tagging facility integrated with Flickr. In *In Proceedings of the AVI 2008 Advanced Visual Interfaces Conference*, pages 326–330. ACM Press, 2008.

[5] Robin Clark. Fast zone discrimination. In *Proceedings of the VLL 2007 workshop on Visual Languages and Logic*, pages 41–54, 2007.

[6] Rosario De Chiara, Ugo Erra, and Vittorio Scarano. VennFS: a Venn Diagram File Manager. In *Proc. of the Seventh International Conference on Information Visualisation, IV 2003, 16-18 July 2003, London, UK*, pages 120–126. IEEE Computer Society Press, 2003.

[7] J. Flower, A. Fish, and J. Howse. Euler diagram generation. *Journal of Visual Languages and Computing*, 19:675–694, 2008.

[8] J. Howse, G. Stapleton, and J. Taylor. Spider diagrams. *LMS Journal of Computation and Mathematics*, 8:145–194, 2005.

[9] Adam Mathes. Folksonomies - cooperative classification and communication through shared metadata. Technical report, 2004. http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html.

[10] Ruth Rosenholtz, Yuanzhen Li, and Lisa Nakano. Measuring visual clutter. *Journal of Vision*, 7(2):17:1–22, 2007.

[11] F. Ruskey. A survey of Venn diagrams. *Electronic Journal of Combinatorics*, 1997. www.combinatorics.org/Surveys/ds5/VennEJC.html.