Fully Automatic Visualisation of Overlapping Sets

Paolo Simonetto, David Auber, Daniel Archambault

Eurographics/ IEEE-VGTC Symposium on Visualization 2009
H.-C. Hege, I. Hotz, and T. Munzner
(Guest Editors)

Volume 28 (2009), Number 3

Fully Automatic Visualisation of Overlapping Sets

Paolo Simonetto, David Auber, and Daniel Archambault

LaBRI, Université Bordeaux 1 & Gravité, INRIA Sud-Ouest, France

Abstract

Visualisation of taxonomies and sets has recently become an active area of research. Many application fields now require more than a strict classification of elements into a hierarchy tree. Euler diagrams, one of the most natural ways of depicting intersecting sets, may provide a solution to these problems.
In this paper, we present an approach for the automatic generation of Euler-like diagrams. This algorithm differs from previous approaches in that it has no undrawable instances of input, allowing it to be used in systems where the output is always required. We also improve the readability of Euler diagrams through the use of Bézier curves and transparent coloured textures. Our approach has been implemented using the Tulip platform. Both the source and executable program used to generate the results are freely available.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms

1. Introduction

Visualisation of hierarchies, usually described by a set of elements and a cluster tree that groups the elements recursively, has been widely studied over the last decade. Many techniques such as Treemaps [JS91] have been proposed as interactive solutions for the exploration of this type of data. However, the containment metaphor, where a parent of the hierarchy strictly contains all of its children, is limited when analysing modern datasets. With increasing frequency, these classifications are no longer cluster trees, but, rather, they are taxonomies where an element can belong to multiple groups at the same level of the hierarchy. Thus, the classification of the elements is no longer a cluster tree but rather a cluster DAG or directed acyclic graph.

A cluster DAG can always be transformed into a cluster tree through unfolding, where branches of the DAG are duplicated. Unfolding has been used [KMBG07, SMO*04] for the visualisation of company relationships and metabolic networks respectively. However, these works demonstrated that in practical cases the number of duplications can be very high. These duplications result in a visualisation that is far more complicated than it needs to be. Approaches, which are not based on data duplication, can be developed once techniques for visualising sets and their intersections are available. Approaches based on Euler diagrams seem to

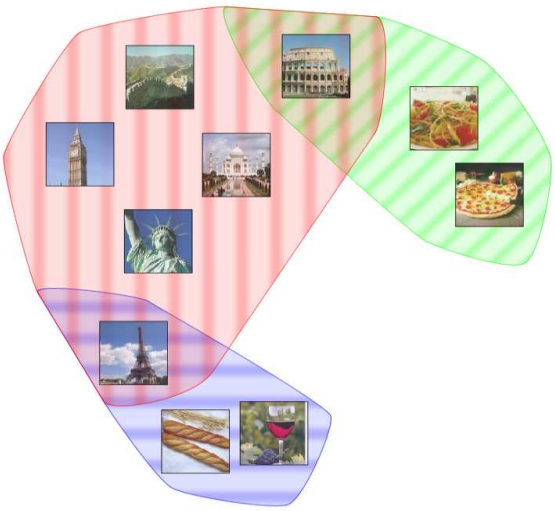


Figure 1: *Example of an Euler diagram rendered with our approach. The red set contains world monuments, the green set contains things that are typically Italian, and the blue set contains things that are typically French. The red set shares the element "Colosseum" with the green set and the element "Eiffel Tower" with the blue set as demonstrated through overlaps of the regions.*

be promising candidates for solutions to these types of problems.

In this paper, we present an algorithm for the automatic generation of Euler diagrams. Our technique is fully automatic, and, unlike previous work, does not have any undrawable instances of input sets and their intersections. This visualisation technique could be used whenever it is necessary to deal with overlapping sets of elements, like in social [PDFV05] or biological networks [BH03], complex query display [VV04], engineering diagrams [FH02] and many other application areas.

2. Related work

Euler diagrams [Eul61] are graphical representations that depict sets and their intersections. In an Euler diagram, a set is a region of the plane bounded by a curve and intersections between sets are depicted through overlaps between these regions as shown in Figure 1. Unfortunately, there is no widely accepted definition of an Euler diagram. Many definitions exist and differ substantially from each other in the types Euler diagrams they can draw [FS, FS06, SRHT]. A variety of algorithms exist using different definitions for an Euler diagram, each with its own set of undrawable instances [Cho07, FH02, FFH08, VV04]. However, all these algorithms suffer from undrawable instances, where the algorithm is unable to produce output for some collections of sets.

The problem of generating **Euler-like** diagrams for any collection of sets and their intersections is a relatively new topic. An Euler-like diagram generalises the notion of an Euler diagram to include disconnected sets, sets with holes, and other constructions which are usually not permitted. Since we focus on Euler-like diagrams for the rest of the paper, we use will almost always use the term Euler diagram when describing our algorithm to mean Euler-like diagram. Simonetto and Auber [SA08] analysed the conditions which resulted in instances of undrawable Euler diagrams. They demonstrated that allowing holes and disconnected sets was sufficient to resolve all undrawable input instances. However, this theoretical work did not present techniques for visualising Euler-like diagrams, rather the results were drawn by hand. Subsequently, the authors presented an algorithm to resolve undrawable input instances and compute set adjacency [SA09], but the work did not present an algorithm for drawing the resultant diagrams.

To our knowledge, the only existing algorithm that generate Euler-like diagrams for every input instance has been developed by Rodgers, Zhang, and Fish [RZF08, RZSF08] as an extension of the algorithm of Flower, Fish, and Howse [FFH08]. However, their approach differ from the one we are presenting for several reasons.

First of all, Rodgers *et al.* explicitly tries to avoid special cases, such as multiple crossing points and concurrent

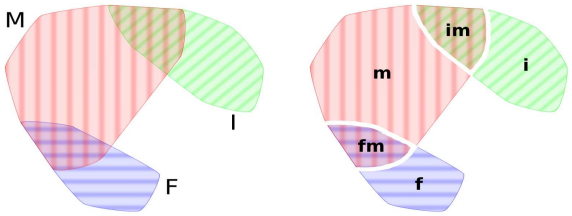


Figure 2: *Classes, zones, and labelling conventions. (a) We use capital letters to denote classes. In this example, M denotes the set of monuments, I the set of Italian things, and F the set of French ones. (b) We use sequences of lower case letters to label zones. For example, the intersection between I and M is labelled im.*

boundaries. In our method, textures and smooth bounding curves for each set mostly handle these cases, as shown in Figure 1, making our approach easier to implement.

Both approaches subsequently improve the initial drawing of the diagram by using a force-directed layout algorithm. However, in our approach, a structure called grid graph is used as an approximation for the set boundaries. This approximation aims to be a good starting point for the layout stage, rather than being visually similar to the final boundaries. This simplifies the process and leads to a clearer division of the stages. Also, it is important to note that adding edge-node repulsive forces to the model does not guarantee that no new crossings will be incurred [Ber99]. In our approach, we use a force-directed algorithm that guarantees nodes and edges will not cross during iterations.

Finally, by inserting set elements before executing the force-directed algorithm, we can produce diagrams where the size of a set is proportional to its cardinality, making better use of space in the plane.

2.1. Definitions and Algorithm Overview

Before we present our approach, we should review some concepts formally defined in previous work. Given a subset S of the sets in the Euler diagram, a **zone** corresponds to the region in the plane where all and only the elements of S intersect. Examples of zones are shown in Figure 2(b). A zone is also the subset of elements contained by region. **Classes**, defined previously in Simonetto and Auber [SA08], are the sets depicted in the drawing of the Euler diagram. Specific classes are denoted with capital letters, and zones are denoted using letter combinations as shown in Figure 2.

At a high-level, the approaches used by previous algorithms are relatively similar. All of the approaches identify three steps as illustrated in Figure 3:

1. *Construct an intersection graph*: construct a planar graph

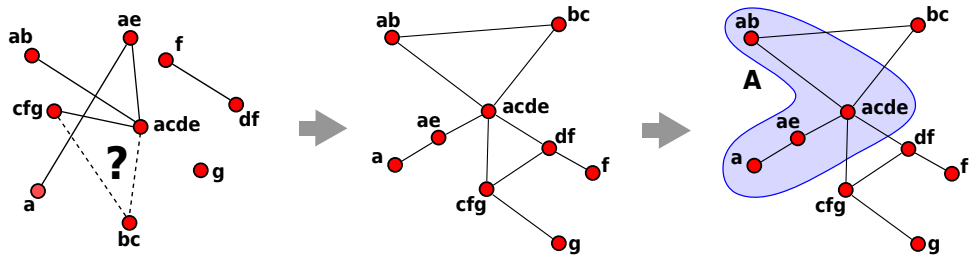


Figure 3: *High-level procedure for drawing Euler diagrams. First, construct the intersection graph. Second, draw the intersection graph using a planar graph drawing algorithm. Third, draw the set boundaries around all nodes in the same class.*

that represents the structure of the Euler diagram, associating nodes of the graph with zones, and selecting edges between zones that should be adjacent.
2. *Draw the intersection graph*: compute a planar drawing of the intersection graph
3. *Construct the set contours*: compute the contours of the classes using the drawing of the intersection graph.

3. Algorithm

The input to our algorithm consists of a listing of the classes in set notation. For example, the input to generate the diagram shown in Figure 1 would be:

Monuments = *{Eiffel Tower, Colosseum, Taj Mahal, Big Ben, Great Wall of China, Statue of Liberty}*
France = *{Wine, Baguette, Eiffel Tower}*
Italy = *{Colosseum, Pasta, Pizza}*

The heuristic of Simonetto and Auber [SA09] is used to generate the intersection graph from the input sets. In this work, the authors describe a greedy approach that ranks all intersection edges with a metric. This metric encodes the contribution the edge would make in creating a well-formed intersection graph. At each iteration, the edge with the highest metric value is inserted into the graph in a way that is similar that Kruskal's algorithm for computing a minimum spanning tree. The algorithm terminates when no more edges can be inserted, either because they will create an intersection graph that is not planar or because they do not significantly contribute to intersection graph structure.

In the heuristic, the metric was developed to encourage the creation of intersection graphs that correspond to a proper Euler diagram whenever possible, i.e. Euler diagrams without disconnected sets. Moreover, the algorithm drives the selection of graph edges toward more readable Euler diagrams. For example, the metric discourages the adjacency of unrelated sets and encourages the adjacency of zones that share a larger number of common sets.

As this paper focuses on the visualisation of overlapping sets, the remainder of this paper concerns itself with drawing Euler diagrams. We accomplish this in four steps:

1. Construct a planar drawing of the intersection graph
2. Build a grid graph around the intersection graph, mapping nodes of the intersection graph to zones.
3. Insert the elements into their zones and apply iterations of *PrEd* [Ber99] to refine the drawing.
4. Select colours and textures for the classes and use Bézier curves to draw the class boundaries.

In the following sections, these four steps are described in detail.

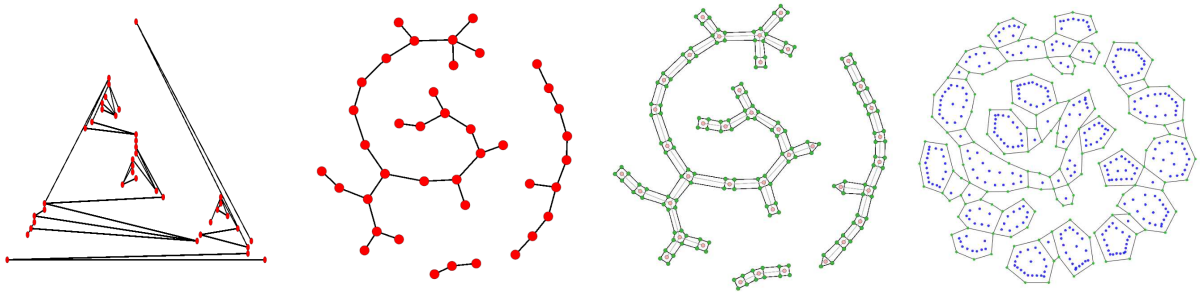
3.1. Drawing the Intersection Graph

Recall that each node of the intersection graph represents a zone, and an edge links two nodes if their zones are adjacent. Therefore, a drawing of the intersection graph decides positions and shapes of the zones and classes in the Euler diagram. Our drawing of the intersection graph must be planar and as regular as possible, avoiding large variations in edge length or angular resolution.

To obtain an initial planar drawing of the intersection graph, we use the algorithm of De Fraysseix *et. al* [DFPP90]. The output of this algorithm is guaranteed to be planar but is quite unsatisfactory with respect to the criteria described above as shown in Figure 4(a). To overcome this problem, we apply the *PrEd* [Ber99] force-directed algorithm. In PrEd, no edge crossings are incurred or eliminated. As there are no crossings in the initial layout, the resulting layout will also be planar. The drawing will generally conform more closely with the above criteria as shown in Figure 4(b). Applying *PrEd* at this stage of the algorithm is not strictly necessary, but it allows for faster convergence and helps the approach avoid local minima.

3.2. Building the Grid Graph

A node of the intersection graph corresponds to a zone of the Euler diagram. All previous work computes the zones im-



(a) (b) (c) (d)

Figure 4: *Overview of our drawing algorithm. (a) Initial output of the FPP algorithm. This layout of the intersection graph is unsatisfactory in appearance. (b) The algorithm draws the intersection graph with* PrEd *[Ber99]. (c) The algorithm constructs a grid graph around the layout of the intersection graph. (d) Elements of the sets are added to the drawing, and the algorithm runs several iterations of* PrEd*. In the four subfigures, nodes of the intersection graph are red, the nodes of the grid graph are green, and the elements of the sets are blue.*

plicitly by tracing the contours around an initial layout of the intersection graph. In our approach, the mapping between a node of the intersection graph and a region of the Euler diagram is explicitly done through a second graph called the **grid graph**. The result of a grid graph computation is shown in Figure 4(c).

The grid graph encloses each node and edge of the intersection graph in its own non-overlapping region of the plane. Thus, by building the grid graph, we will identify regions of empty space around nodes and edges of the intersection graph. We call these regions **node-regions** and **edge-regions** respectively. As long as these regions are of limited size, zone overlaps that do not correspond to set intersections are avoided through a planar drawing of the intersection graph.

To construct the grid graph, we start by computing a set of non-intersecting circles around the nodes of the intersection graph. Each circle centred on a corresponding node in the intersection graph layout and will be used to define the node-regions and the edge-regions of the grid graph directly.

3.2.1. Computing the Set of Circles

The position of each circle is given by the position of each node in the layout of the intersection graph. Thus, we can choose a single radius R for all circles that is just small enough to avoid all overlaps. If we inscribe our node regions on the interior of these circles, we are guaranteed that the node and edge regions of the grid graph do not overlap.

For these reasons, we choose an R which is less than half the minimum node-to-node and edge-to-node distance. In our system, we use:

$$R = \min\left(\frac{\min\limits_{v,w\in V, v\neq w} d(v,w)}{3}, \frac{\min\limits_{v\in V, e=\{w,z\}\in E, v\neq w,z} d(v,e)}{3}\right)$$

where $I=(V,E)$ is the intersection graph and d is a function which defines the distance between elements.

3.2.2. Constructing the Node-Regions

We will now inscribe a polygonal region on the interior of each circle, defining the node-regions. To guarantee that the centre of the circle is inside the polygonal region, adjacent polygon vertices in a anticlockwise traversal form a maximal central angle of $2\pi/3$ denoted α.

Let us consider a circle centred on each node of the intersection graph. Each edge of the intersection graph divides the circle into several sectors. More formally, consider the set $G \subseteq E$ of edges incident to a node $v \in V$ of an intersection graph $I=(V,E)$. As the layout of the intersection graph is fixed, we can consider each edge $e_j=(v,w_j)$ of G as a vector $\mathbf{e_j}=\overrightarrow{vw_j}$. If we sort the $|G|$ vectors by increasing angle from the x-axis into the ordered sequence $\mathbf{e}_1,\mathbf{e}_2,\ldots\mathbf{e}_k,\mathbf{e}_{k+1}$, $\mathbf{e}_1=\mathbf{e}_{k+1}$, adjacent elements define sectors around v. When v has no incident edges, we will consider the entire circle as a single sector.

For each sector $(\mathbf{e}_i,\mathbf{e}_{i+1})$, we select a set of $\lceil\gamma/\alpha\rceil$ polygon vertices where γ is the central angle of the sector. The polygon vertices are evenly spaced δ radians apart on the circumference of the circle. The process is shown in Figure 5.

3.2.3. Constructing the Edge-Regions

When constructing the node-regions, the circle around each node of the intersection graph is divided into ordered sectors. Thus, each edge of the intersection graph is a boundary of an ordered sector. Given a node of the intersection graph $n \in V$ and edge of the intersection graph $e \in E$, with e incident to n, we will denote $c(n,e)$ and $a(n,e)$ as the rays emanating from n that bound e on clockwise and anticlockwise traversals of the polygon vertices respectively.

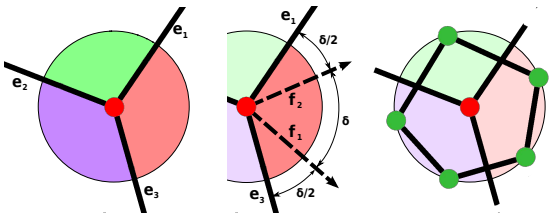


Figure 5: *Constructing a node-region. (a) The edges incident to the node of the intersection graph define three sectors, bounded by* $(\mathbf{e}_1,\mathbf{e}_2)$, $(\mathbf{e}_2,\mathbf{e}_3)$, *and* $(\mathbf{e}_3,\mathbf{e}_1)$. *(b) Considering only the red sector, we are able to define the vectors* $\mathbf{f}_1,\mathbf{f}_2$ δ *radians apart. (c) The intersections between the vectors* $\mathbf{f}_i$ *and the circumference of the circle are taken as vertices of the node-region.*

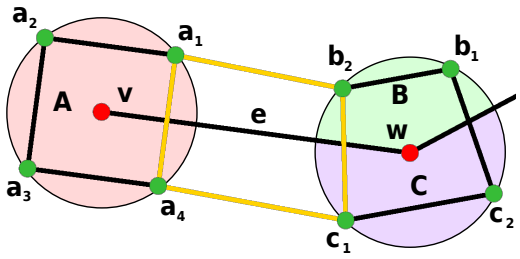


Figure 6: *Construction of an edge-region. An edge of the intersection graph divides the circles into the sectors A, B and C. The polygon vertices computed in the previous step are ordered clockwise and anticlockwise. In the diagram, the edges of the grid graph which define the edge-region of e are:* (a_4,a_1), (b_2,c_1), (a_4,c_1), *and* (b_2,a_1).

Let $p_{c(n,e)}$ and $p_{a(n,e)}$ be the polygon vertices supporting these rays. The edge-region for the edge $e=(v,w)$ of the intersection graph is therefore bounded by the following four edges of the grid graph: $(p_{a(v,e)},p_{c(v,e)})$, $(p_{a(w,e)},p_{c(w,e)})$, $(p_{a(v,e)},p_{c(w,e)})$, and $(p_{a(w,e)},p_{c(v,e)})$. An example of an edge region is shown in fig. 6.

3.3. Element Insertion and Boundary Refinement

Once an initial planar layout of the grid graph is defined, we will run *PrEd* to refine the drawing. But before we run it a second time, we insert the elements inside the classes and compute the edges that will be used in the drawing of the class boundaries. Figure 4(d) shows the result after this step.

Inserting elements into their respective zones is relatively straightforward. Each zone in the grid graph is bounded by a polygon inscribed on the interior of a circle of radius R and the maximum permissible central angle is $2\pi/3$. Thus, the closest a polygon edge can ever be to the centre of the circle is $R/\sqrt{3}$. Thus, the set elements are randomly placed inside a circle of this radius.

To determine which edges of the grid graph define a class boundary, we have to consider subgraphs of the intersection graph or class schema as defined precisely in Simonetto and Auber [SA08]. A **class schema** is an induced subgraph on the intersection graph such that it includes all the nodes belonging to the same class. The grid graph edges that define a class are the edges that enclose the schema without crossing it. If we consider the edge (v,w) in Figure 6 as its own class, then the class schema will be formed just by v, w and e. We can therefore remove the grid graph edges (a_4,a_1) and (b_2,c_1), because they intersect e. Of course these grid graph edges may be used by other classes, so we must consider all classes before deleting these edges.

3.4. Euler Diagram Rendering

Many Euler diagram generation algorithms stop after the class boundaries have been identified by closed curves, and elements are placed inside their appropriate zones. However, as our drawings are intended for visualisation tasks, we should pay close attention the rendering of the diagram. In this section, we discuss how we select colours, textures, and bounding curves for classes.

3.4.1. Colour Selection

Humans have trouble distinguishing more than about six to eight colours [War00] in a complex diagram. If we use transparency and allow colours mix at each intersection, we can see why minimising the number of colours used becomes a priority. On the other hand, different colours should be used for different classes in the diagram.

To circumvent this problem, we generate another graph where each node is a class of the diagram and an edge indicates that two classes overlap. We run the node colouring heuristic of Welsh and Powell [WP67] on this graph. The output of this algorithm suggests a small number of colours and textures that can be assigned to the classes of the diagram. Assuming that the graph can be coloured with less than c colours and we have c colours and textures at our disposal, it is guaranteed that no two sets which overlap will be assigned the same colour and texture combination. Currently, we use $c=8$.

3.4.2. Textures

We implemented the work of Byelas and Telea [BT08] to select textures regions of the classes overlap. By using textures in addition to colour, we are better able to represent regions more distinctly than by simply using colours or class boundaries alone. Intersection patterns between the textures also can help indicate the classes to which a zone belongs and helps the approach scale to larger Euler diagrams.

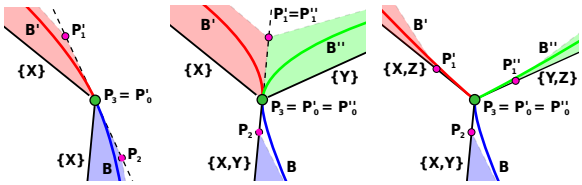


Figure 7: *Construction of smooth Bézier curves at grid graph vertices. (a) Case 1: if two incident edges e_1 and e_2 of the grid graph share the same set of classes, the orientation of l is given by the line perpendicular to the bisector of the angle between e_1 and e_2. (b) Case 2: if three or more incident edges, e_i, $i = 1 \ldots n$, are associated with different sets of classes, and there exists an edge e_k that is the union of all these classes, the orientation of l is the orientation of e_k (c) Case 3: if neither of the above-cases apply, we cannot have continuous Bézier curves at this grid graph vertex. Thus, we place the control points along their associated grid graph edges. That way, no new boundary intersections are introduced.*

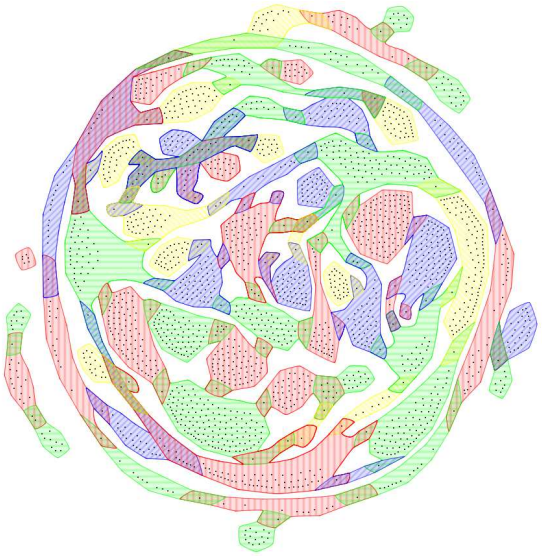


Figure 8: *A diagram showing sixty highest rated movies in IMDB with full credited cast.*

3.4.3. Bounding Curves

We convert the polygonal boundaries of classes into continuous Bézier curves, making them easier to follow. We transform each edge of the grid graph into its own Bézier curve, respecting the following properties:

- do not introduce any new set boundary intersections or change inclusion or exclusion of the set elements
- smoothly join Bézier curves at grid graph vertices

The curve drawn for each edge $e = (v,w)$ has a control polygon of four control points: $P_0, \ldots P_3$. The points P_0 and P_3 are always coincident with the nodes v and w. The position of v is also used to calculate the position of P_1 and the position of w is also used to calculate the position of P_2. For this reason, we say P_1 is related to v and P_2 is related to w.

In order to respect all the above conditions, we must choose the positions of P_1 and P_2 in such a way that:

- no control polygon overlaps with another control polygon or set element
- when two control polygon edges meet at a common grid graph vertex, they must be collinear along the line l

The line l can have any orientation in the plane. The orientation of l is calculated individually for each grid graph vertex in the following way as shown in Figure 7:

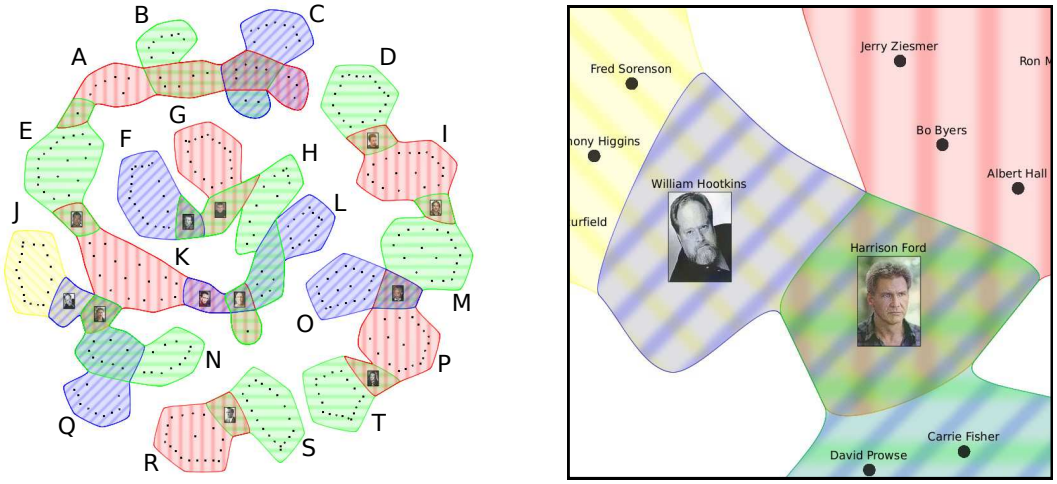
1. if two incident edges e_1 and e_2 of the grid graph share the same set of classes, the orientation of l is given by the line perpendicular to the bisector of the angle between e_1 and e_2
2. if three or more incident edges, e_i, $i = 1 \ldots n$, are associated with different sets of classes, and there exists an edge e_k that is the union of all these classes, the orientation of l is the orientation of e_k
3. if neither of the above-cases apply, we cannot have continuous Bézier curves at this grid graph vertex. Thus, we place the control points along their associated grid graph edges. That way, no new boundary intersections are introduced

We calculate the maximum displacement of each control point along l that does not introduce any new crossings. This displacement is calculated using a function of PrEd [Ber99].

4. Results on Real World Data

We tested our approach on a few real world datasets. The Internet Movie Database (IMDB) is an online database of information related to films, actors, and the cinema. It also ranks films according to viewer ratings.

To show how our approach can scale to larger datasets, Figure 8 is the diagram generated for sixty of the top seventy films of the database with full credited casts for each film. All the discarded films were non-overlapping. The diagram categorises over two thousand actors. Although the diagram is complex, it is still fairly readable. Subsequently, we selected a subset of twenty films from the top forty highest ranked films. For each film, we considered the first twenty actors in credits order. Figure 9 shows how the films, the sets in the diagram share cast members. In this diagram, we can make a few interesting discoveries. First of all, there are three set of classes, labelled $\{A, B, C\}$, $\{N, Q\}$ and $\{H, L\}$, that have extensive overlaps. These films are actually film trilogies: “The Lord of the Rings”, “Star Wars” and “The



(a) (b)

Figure 9: *An Euler diagram of IMDB movies composed by about twenty movies. (b) An overall look. The sets have been labelled with capital letters. A - The Lord of the Rings - Part II (2002). B - The Lord of the Rings - Part I (2001). C - The Lord of the Rings - Part III (2003). D - American History X (1998). E - The Matrix (1999). F - Taxi Driver (1976). G - Goodfellas (1990). H - The Godfather: Part II (1974). I - Fight Club (1999). J - Raiders of the Lost Ark (1981). K - Apocalypse Now (1979). L - The Godfather (1972). M - Se7en (1995). N - Star Wars: Episode V (1980). O - The Shawshank Redemption (1994). P - The Dark Knight (2008). Q - Star Wars (1977). R - The Usual Suspects (1995). S - American Beauty (1999). T - Léon (1994). Capital letters labelling sets are placed manually, but every other part of the diagram generation was automatic. (b) A close-up of part of the diagram. In these figures, images of the actors are shown when they are available in the directory.*

Godfather". It makes these sets of films would share many actors in common. Secondly, it's also interesting to see that stars, such as Harrison Ford, Marlon Brando, Robert De Niro, Brad Pitt and Morgan Freeman naturally stand out in the diagram. They are contained in the intersection of many classes with other elements. Finally, it is important to note that textures greatly improve the readability of the diagram. In Figure 9(b), we can distinguish the many classes that are sharing the zone of Harrison Ford, even though the boundaries of the classes are not clearly visible.

For our second example, shown in Figure 10, we show the full credited cast of the "The Lord of the Rings" trilogy. First off, all the major protagonists are in a single zone as shown in Figure 10(a) and in the close-up in Figure 10(b). This zone corresponds to the intersection of the three movies. Miranda Otto, the actor who plays Eowyn, appears only in "The Two Towers" and "Return of the King", is an exception in the upper left corner of the figure. We can also see that Billy and Katie Jackson, the children of the director, Peter Jackson, made cameo appearances in all three films.

5. Future work

Our algorithm for drawing Euler-like diagrams can be improved in many ways. As seen in Figure 11, it is often possible to compute better shapes for classes. The best way to get well formed contours more consistently would be to improve the force-directed algorithm stages of our system. We would also like to improve the execution and convergence speed of PrEd. As PrEd comprises nearly 95% of the computation time, speeding up this algorithm would greatly improve the

(a) (b)

Figure 10: *Cast of the trilogy "The Lord of the Rings". (a) The diagram generated. "The Fellowship of the Ring" is red, "The Two Towers" is green, and "The Return of the King" is blue. (b) The zone shared by all three films. In these figures, images of the actors are shown when they are available in the directory.*

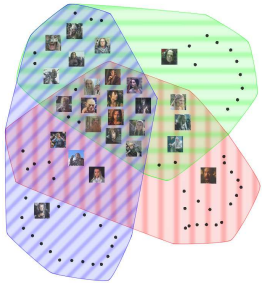


Figure 11: *Direction for future work. The original diagram can be improved simplifying the shape of the curves. This mock-up was created with a version of the algorithm that is under development. As seen in this picture, at the moment we cannot guarantee that new regions will not be created.*

execution speed of our system and allow us to scale to even larger datasets.

In this work, we focused primarily on algorithmic contributions required to draw collections of sets. There are many other problems associated with the visualisation of sets that have yet to be explored: interactive manipulation of sets, how sets could be extended into multi-level settings, and the visualisation of dynamically changing sets. Moreover, we should investigate how this technique could be applied to applications in the humanities, sciences, social sciences, as well as many other fields. More specifically, we would like to apply this technique to the problem of visualising overlapping graph clusters, which arise in pathway visualisation of metabolic networks or protein complexes in protein-protein interaction networks.

6. Conclusions

Our work provides a fully automatic method for the generation of Euler-like diagrams. The system produces a drawing for any collection of input sets. It uses the work of Simonetto and Auber [SA08, SA09] as a basis. In order to circumvent undrawable instances, it must disconnect regions or introduce holes into them. Our approach easier to implement than other competitive previous work, because it does not try to prevent collinear boundaries. By using colours and textures more effectively, we are no longer required to solely rely on contours, allowing us to scale to larger dataset sizes.

Acknowledgements

This research is partially funded by ANR-BBSRC Systryp project and the INRIA Gravité project.

References

[Ber99] BERTAULT F.: A force-directed algorithm that preserves edge crossing properties. In *Graph Drawing* (1999), Lecture Notes in Computer Science, Springer, pp. 351–358.

[BH03] BADER G. D., HOGUE C. W.: An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics 4*, 2 (Jan. 13 2003), 27.

[BT08] BYELAS H., TELEA A.: Texture-based visualization of metrics on software architectures. In *SoftVis08. Proceedings of the 4th ACM symposium on Software visuallization* (New York, NY, USA, 2008), ACM, pp. 205–206.

[Cho07] CHOW S. C.: *Generating and drawing area-proportional Euler and Venn diagrams*. PhD thesis, 2007.

[DFPP90] DE FRAYSSEIX H., PACH J., POLLACK R.: How to draw a planar graph on a grid. *Combinatorica 10* (1990), 41–51.

[Eul61] EULER L.: Lettres à une princesse d'allemagne, letters no. 102-108, 1761.

[FFH08] FLOWER J., FISH A., HOWSE J.: Euler diagram generation. *Journal of Visual Languages and Computing 19* (Dec. 2008), 675–694.

[FH02] FLOWER J., HOWSE J.: Generating Euler diagrams. *Lecture Notes in Computer Science 2317* (2002).

[FS] FISH A., STAPLETON G.: Defining Euler diagrams: choices and consequences. *Euler Diagrams 2005*.

[FS06] FISH A., STAPLETON G.: Formal issues in languages based on closed curves. In *Distributed Multimedia Systems, Knowledge Systems Institute* (2006), pp. 161–167.

[JS91] JOHNSON B., SHNEIDERMAN B.: Treemaps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. of the 2nd International IEEE Visualization Conference* (1991), pp. 284–291.

[KMBG07] KOENIG P.-Y., MELANÇON G., BOHAN C., GAUTIER B.: Combining DagMaps and Sugiyama layout for the navigation of hierarchical data. In *IV* (2007), IEEE Computer Society, pp. 447–452.

[PDFV05] PALLA G., DERÉNYI I., FARKAS I., VICSEK T.: Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435* (2005), 814–818.

[RZF08] RODGERS P., ZHANG L., FISH A.: General Euler diagram generation. vol. 5223, Springer, pp. 13–27.

[RZSF08] RODGERS P., ZHANG L., STAPLETON G., FISH A.: Embedding wellformed Euler diagrams. *12th International Conference on Information Visualisation* (2008), 585–593.

[SA08] SIMONETTO P., AUBER D.: Visualise undrawable Euler diagrams. In *IV08* (July 2008), IEEE Computer Society, pp. 594–599.

[SA09] SIMONETTO P., AUBER D.: An heuristic for the construction of intersection graphs. In *IV09* (July 2009), IEEE Computer Society.

[SMO*04] SHANNON P., MARKIEL A., OZIER O., BALIGA N. S., WANG J. T., RAMAGE D., AMIN N., SCHWIKOWSKI B., IDEKER T.: Cytoscape: A software environment for integrated models of biomolecular interaction networks.

[SRHT] STAPLETON G., RODGERS P., HOWSE J., TAYLOR J.: Properties of Euler diagrams. *Electronic Communications of the EASST*.

[VV04] VERROUST A., VIAUD M.-L.: Ensuring the drawability of extended Euler diagrams for up to 8 sets. In *Diagrams 2004, 3rd International Conference.* (2004), vol. 2980 of *Lecture Notes in Computer Science*, Springer, pp. 128–141.

[War00] WARE C.: *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, San Francisco, 2000.

[WP67] WELSH D., POWELL M.: An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal 10*, 1 (1967), 85–86.