

An Heuristic for the Construction of Intersection Graphs

Paolo Simonetto, David Auber
LaBRI, Université Bordeaux I and Gravit , Inria Sud-Ouest
paolo.simonetto@labri.fr, auber@labri.fr

February, 2009

Abstract

Most methods for generating Euler diagrams describe the detection of the general structure of the final drawing as the first step. This information is generally encoded using a graph, where nodes are the regions to be represented and edges represent adjacency. A planar drawing of this graph will then indicate how to draw the sets in order to depict all the set intersections.

In this paper we present an heuristic to construct this structure, the intersection graph. The final Euler diagram can be constructed by drawing the sets boundaries around the nodes of the intersection graph, either manually or automatically.

Keywords—Euler diagrams, overlapping clustering

1 Introduction

Euler diagrams were introduced a long time ago [3] and have been widely used in many fields, but their automatic generation is a quite recent topic. There may be several explanations for this.

First of all, there is no universally accepted definition of a Euler diagram. Euler diagrams were introduced informally, and scientists and mathematicians have made different interpretations of the characteristics they have.

Secondly, Euler diagrams are not always drawable. All the existing definitions of Euler diagrams have sets that can be supplied as input but cannot be represented. The exact class of non-representable instances clearly depends on the definition, but the set is never empty.

Thirdly, Euler diagrams are most useful when they are clear and visually appealing. The definition of these aspects is hard to quantify and it is even more difficult to create these conditions on the final drawing.

Recent studies on automatically drawing Euler diagrams have been facing these problems. The lack of a strict definition has initiated the analysis of several kinds of Euler or Euler-like¹ diagrams: Chow [2] dealt with

¹Classes of diagrams that extend the most accepted definitions are generally referred as Euler-like.

the most common definition, Flower and Howse [6] with a restricted version, while Verroust and Viaud [11] and Simonetto and Auber [9] with two extended versions.

Fish, Stapleton *et al.* [4, 5, 10], on the other hand, studied the differences between the many definitions and described how the characteristics enforced (for instance admitting or denying multiple curve crossing points or concurrent boundaries) influence the class of the representable instances.

Moreover, Benoy and Rodgers [1] started their user study on the comprehension of Euler diagrams, analysing how easily people can understand a diagram under different aesthetic conditions.

It is finally interesting to notice how Euler diagrams have not been studied only as an abstract concept, but as a way to answer really concrete visualisation problems. The previously cited contributions came from a variety of fields spanning software engineering diagrams [4, 5, 10, 6], video database queries [11], and methods for visualising overlapping clusters on graphs [9].

The intuitiveness of Euler diagrams and their wide range of applications show how a general method, applicable to all the possible input cases, would answer many visualisation problems.

In this paper, we present an algorithm to construct a graph that represents the skeleton of *Euler representations*, a class of always drawable Euler-like diagrams. As for most of the other approaches to the generation of Euler diagrams, this represent a first and crucial step for the depiction of the final structure.

2 Related work and definitions

The generation of Euler diagrams has been studied according to the different interpretations on Euler diagrams [2, 6, 11]. Unfortunately, all these methods suffer of undrawable instances, which are set systems not representable with the diagrams of the class.

Undrawable instances exist because unrelated sets should not overlap. In fact, it might be impossible to draw a set without overlapping at least another unrelated one,

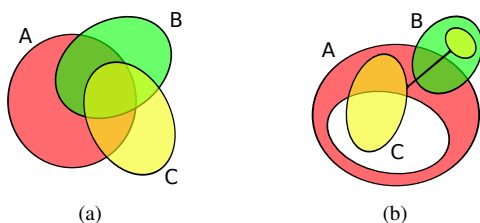


Figure 1: Euler and Euler-like diagrams. (a) an Euler diagram according to the most accepted definition, reported in [2]. (b) the Euler representation described in [9]. We can see that the set C is not formed by a single connected region and that set A has a “hole”. When a set is disconnected, a link is put to connect the separate regions.

and so to create a proper Euler diagram. That conflicts with the idea of providing an output for every possible input.

To our knowledge, the only other approach able to deal with every input is the one described by Rodgers, Zhang and Fish [7] extending the methods proposed in [6, 8]. However, the authors did not concentrate on the construction of the structure graph, particularly when the diagram cannot be represented in a planar way. For this reason, we investigated the issue further.

Euler representations. In a previous paper [9], we analysed the conditions necessary to avoid undrawable instances in Euler-like diagrams. The study resulted in a class of Euler-like diagrams that:

- might contain holes inside the main region of a set. In other words, a set might not be bounded by exactly one closed line.
- might have disconnected set regions. This means that a set might be represented with two regions that may be far from each other.

The ambiguities introduced by these two visual structures could be overcome by colouring the actual set region and by introducing links between the disconnected set regions. This leads to diagrams similar to the one showed in figure 1(b), called *Euler representations*.

Additional methods for improving the understanding of complex Euler representations have also been suggested. However, as these cases find a proper application only in very rare and complex cases, they are not taken into consideration in this paper.

Classes and zones. In the paper, the sets to be drawn are called *classes* to avoid confusion with the more common word “set”. They are indicated with capital letters.

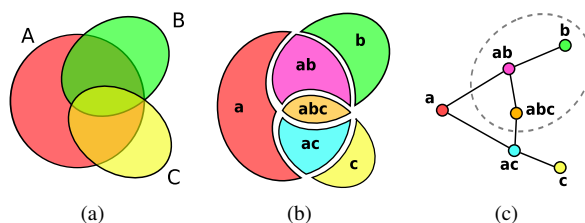


Figure 2: Transformations between Euler diagrams and intersection graphs. (a) the original diagram. (b) detection of the diagram zones. (c) the resulting intersection graph. The dashed line is not part of the graph, but shows how to reverse the procedure. For drawing the boundary of the class B we need to enclose the nodes b, ab, abc and intersect the edges $(a, ab), (ac, abc)$.

The regions that are shared by a subset \mathcal{S} of classes and are external to all the other existing classes are defined as *zone*. They are indicated with lower case sequences of letters, corresponding to the elements of \mathcal{S} .² Examples of zones are graphically shown in figure 2(b): there we can see how zone ab is formed by the intersection of the classes in the subset $\mathcal{S} = \{A, B\}$, minus the region that is also inside C .

When referring to zones, we indicate a generic sequence of lower case letters with a Greek letter. For instance we might have that $\alpha = abd$ or $\beta = ac$. We use the symbol $*$ to indicate all the zones sharing the same letters. For instance, the notation $ab*$ indicates the set of all the zones which labels contain ab .

Finally, the function \mathcal{C} returns the subset of classes used to generate the zone. For instance $\mathcal{C}(ab) = \{A, B\}$.

Intersection graphs. In [9] we showed how Euler diagram can be generated starting from a graph that summarises its structure, and vice versa. This graph, called *intersection graph*, has nodes corresponding to the zones of the final diagram and edges that link adjacent zones. An example of intersection graph and of the transformations involved is shown in figure 2.

Intersection graph characterisation. The intersection graph describes the connectivity of the final diagram. Thus, many characteristics of the resulting diagram depend on the topology of the intersection graph. We defined the following rules valid for well-formed intersection graphs:

²As the letters in the labels derive by set elements, the actual order in which they appear is not important.

1. The intersection graph must be planar.
2. Each *class schema*, that is the subgraph induced by the nodes of a class (the set of zones a^*, b^*, \dots), should be connected. This avoids the generation of disconnected classes.
3. For each α , each subgraph induced by a set of zones α^* should be connected. This avoids the same classes intersecting in disconnected regions (in figure 3(b), let $\alpha = ab$. The subgraph induced by $\alpha^* = \{ab, abc\}$ is not connected. Therefore, the classes A and B intersect in two separate regions).
4. Each subgraph induced by the zones external to a class (the set of zones $\bar{a}^*, \bar{b}^*, \dots$) should be connected.³ This avoid holes.
5. The intersection graph should have a compact and regular layout.

The first rule must always be respected. In order to build a proper Euler diagram, we have to construct an intersection graph that strictly satisfies also rule number 2 and 4. However, we have no other strict limitations than rule 1 when building the intersection graph of a Euler representation.

3 Method overview

When constructing an intersection graph, we have no flexibility in the nodes to insert: each node corresponds to a non-empty zone. The real problem is identifying which edges to insert.

Even for Euler diagrams, there might be many selections of edges that lead to a correct output. Unfortunately, their clarity might vary greatly, as shown in figure 3. The same problem appears to a greater extent with Euler representations, as we have more freedom in the construction of their intersection graphs.

Thus, the construction of intersection graphs consists of an optimisation procedure that aims to:

- identify a result as similar as possible to a proper Euler diagram,
- choose intersection graph configurations that lead to as clear diagrams as possible.

The approach. Our algorithm constructs the intersection graph inserting one edge at each step. Thus, the main part of the algorithm is constituted by a procedure that at each iteration selects one edge and inserts it. The

³At this step it might be necessary to add a node representing the null zone, that is the area external to every class.

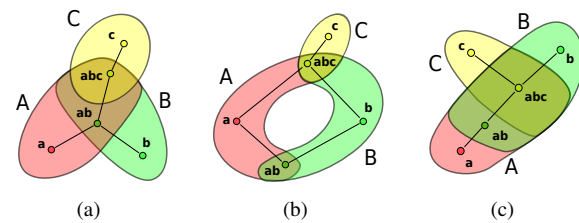


Figure 3: Intersection graphs for three classes A, B, C having the zones a, b, c, ab and abc non-empty. There are many sets of edges that lead to valid intersection graphs; here three of them are shown. The configuration in subfigure (a) is clearer than the others, as in (b) classes A and B intersect in two separate regions and in (c) it is more difficult to distinguish the classes involved in each zone.

selection of the edges is steered by a metric that encode the edge contribution in respecting the rules enounced before. The insertion procedure and the metric are described in more detail in the following sections.

4 The insertion procedure

The algorithm we developed follows a procedure similar to the classical Kruskal's algorithm for minimum spanning trees. We start with a graph that has no edges, and at each step we insert the edge with higher metric value. Once inserted it remains part of the resulting intersection graph.

The metric encodes the rules that characterise a well-formed intersection graph. However, there is a particular condition we prefer to handle at this level, rather than in the metric calculation: the planarity. In fact, as the resultant graph must be planar, this condition should not compete with other aesthetic aspects.

Initialisation We start with a graph that has a node for each non-empty zone, and no edges.

During the computation, we keep a pool of candidate edges for insertion. This collection is initialised with all the possible edges, and it will contain only edges that have not been inserted, discarded, or weighted with negative or null values.

Iterations At each iteration, the following steps are executed:

1. The best edge of the pool is selected.
2. A planarity test is executed to determine whether the edge makes the graph unplanar.

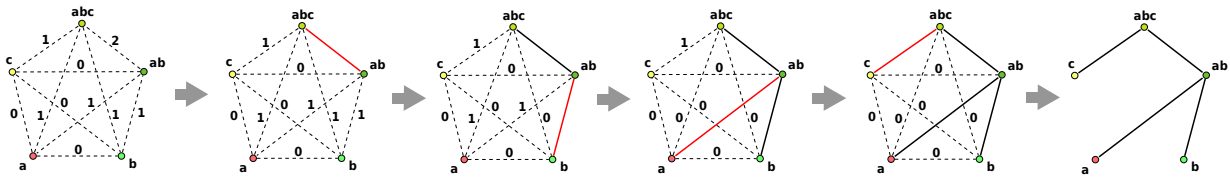


Figure 4: Values of c for the edges of the intersection graph for several edge insertions. At the first step we suppose that the edge (ab, abc) with value 2 (since it would merge components for the class schemas of A and B) is the one selected to be inserted. At the second step, let us suppose we insert the edge (b, ab) with value 1 (it connects the connected components $\{b\}$ and $\{ab, abc\}$ in the class schema B). Because of the insertion of this edge, the edge (b, abc) updated its value to 0. The same happens to the edge (a, abc) after the insertion of (a, ab) .

3. If the graph is no longer planar, the edge is discarded.
4. If the graph is still planar, the edge is added and the weights updated.

The algorithm terminates when there are no more edges in the pool, that indicates no more useful edges are available for the insertion.

The output When the algorithm terminates, we obtain the final intersection graph. Once layed out, the graph will allow us to draw the boundaries of the sets to be represented in order to generate the Euler representation.

The final intersection graph might encode a representation where classes are disconnected regions. This happens when it is impossible to connect zones of the same class without breaking the planarity. This result is perfectly normal since non-planar intersection graphs are the main cause of undrawable Euler diagrams. This is also the reason why Euler representations sometimes use links (a link is shown in figure 1(b), even if in that case it was not necessary).

5 The metric

The weight assigned to the edges encodes how much their insertion in the current intersection graph would contribute in the optimisation procedure. In other words, how much the insertion of an edge would contribute in obtaining a well-formed and readable Euler diagram.

Promote class connections. In order to avoid the insertion of links, we need to insert edges between all the nodes of the same class to make their induced subgraph connected. Hence, this property will be the highest priority in weighting the edges.

Reducing the number of edges is a good idea, since the more edges we need to insert, the more likely the graph violates planarity. We will favour edges that connect

components in many class schemas at the same time, rather than edges that connect few of them.

For this reason, we define the function $c(e)$. The function counts the number of class schemas in which the insertion of e would connect disconnected components. It is worth noting that this function depends on the edges previously inserted. An example of what the function c means and how it works when updating a graph, is shown in figure 4.

Promote aesthetic. In order to obtain connected set intersections and avoid unclear configurations (such as shown in figure 3), we will promote a structure where the zones connected do not differ too much from each other. This is obtained using two functions:

$$u(e) = \min(|\mathcal{C}(s_e)|, |\mathcal{C}(t_e)|) - |\mathcal{C}(s_e) \cap \mathcal{C}(t_e)|$$

$$v(e) = \max(|\mathcal{C}(s_e)|, |\mathcal{C}(t_e)|) - |\mathcal{C}(s_e) \cap \mathcal{C}(t_e)| - 1$$

where e is a candidate edge, and s_e and t_e are the zones corresponding to the two incident nodes.

The first function aims to detect the number of unrelated classes that are forced to be adjacent through this edge. For instance, the edge (ab, ac) forces classes B and C to share a boundary even when it is not necessary. This is not the case for the edge (a, abc) , since no unwanted boundaries would overlap. The function u returns 1 in the first case, but 0 in the second. An example of why it is important to penalise these edges is shown in figure 5.

The second function penalises edges between nodes that differ for more than one class. For instance, the edge (a, abc) connects two zones where the second is included in two classes, B and C , more than the first. When choosing between (a, ab) or (ab, abc) there is just one class more for one of the two nodes. The function v returns 1 in the first case and 0 in the second. Figure 6 shows why this penalty is helpful.

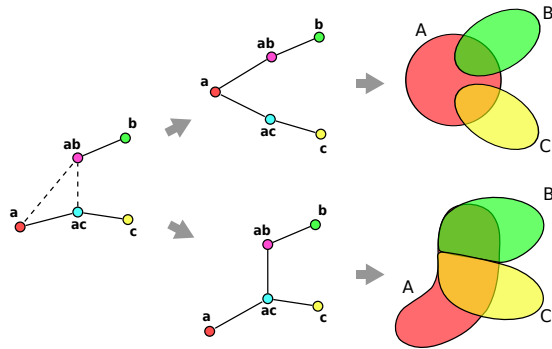


Figure 5: Unwanted proximity. At the current stage of the intersection graph we need to insert the edge (a, ab) or the edge (ab, ac) . The picture shows the result of the two choices. The diagram below is much less clear, as it makes two unrelated classes to share their boundaries.

Concerning holes. The metric does not explicitly promote a final diagram without holes. In fact, the problem can be handled at a different level.

The insertion of a node relative to the null zone, as explained in [9], has two main aims: defining the outer face of an intersection graph embedding and forcing the zones linked to it to stay on the outer face.

In our automatic approach for the generation of Euler representations we answer these issues working directly on the graph embedding. In fact, being able to operate on the embedding and choose the external face in order to maximise the number of outer nodes allowed us to extend the limits of our method.

The final metric. The metric value for each edge e is obtained combining the previous functions:

$$w(e) = c(e) - p_1u(e) - p_2v(e)$$

where the p_1 and p_2 are parameters that define the penalty weights.

When choosing the parameter values, we recommend that they are a small fraction of unity, in order to preserve the central importance of c . In fact, u and v have been developed with the idea that they should almost only influence the rank of edges with the same value of c .

Finally, we suggest $p_1 \geq p_2$, as the condition related to v is less confusing than the one associated to u .

6 Examples

We report some intersection graphs generated with this method. Figure 7 shows two set configurations involving several overlaps and inclusions. The relative Euler diagram can be easily detected by drawing lines that enclose the nodes of each class.

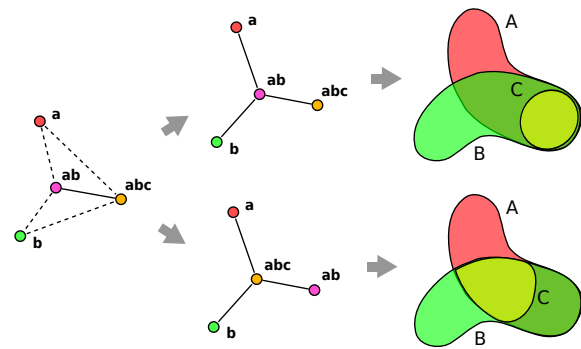


Figure 6: Multiple class difference. We can now decide either to insert the edges (a, ab) and (b, ab) , or the edges (a, abc) and (b, abc) . The figure shows the intersection graphs and the resulting diagrams for both the choices. The first diagram shows more clearly the way the classes overlap.

Figure 8(a) shows the intersection graph obtained on a real world case. We extracted film data from the Internet Movie Database⁴ and we defined 17 classes composed by actors who take part to the same film. We obtained a Euler representation by manually enclosing the nodes with lines and inserting nodes to represent the class elements. The resulting diagram (see figure 8(b)) shows, in an appealing way, the overlaps created by actors who took part in more than one film.

We can also note how the methods detected a proper Euler diagram, as there were no planarity problems. Moreover, that the metric proposed contributed in detecting a clear structure of the intersection graph, as the unclear configuration of figure 3 are avoided.

⁴www.imdb.com

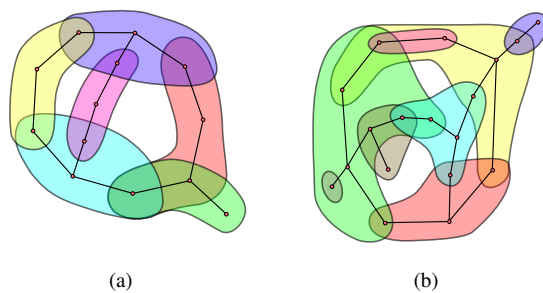


Figure 7: Two intersection graphs generated with our method. The elements of the graph allowed to detect the class boundaries, which have been manually drawn.

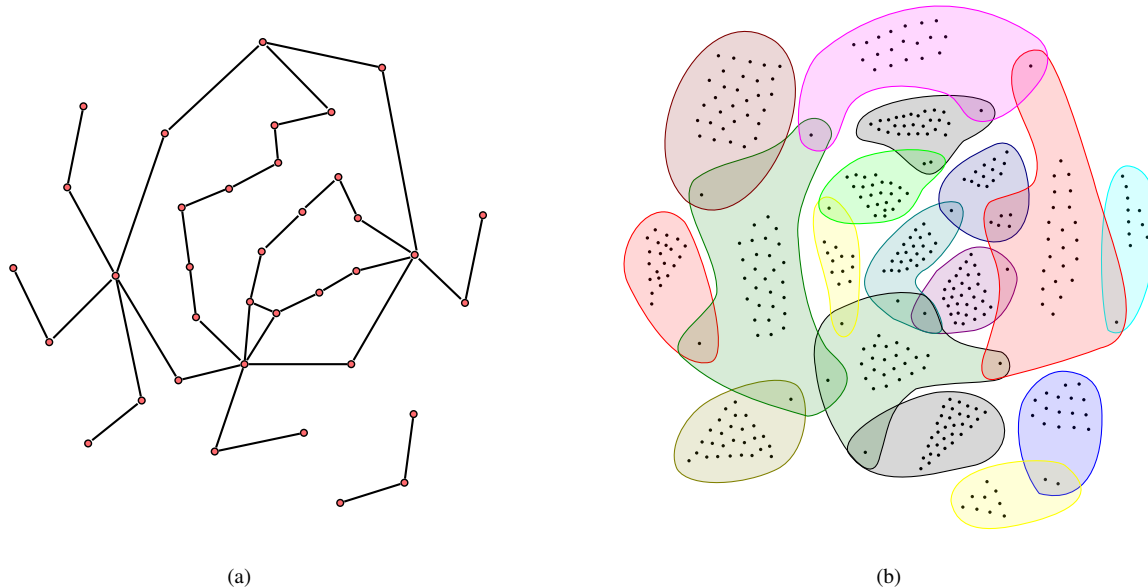


Figure 8: Application of the method to real world data. (a) the intersection graph generated and drawn in a planar way. (b) the Euler diagram manually generated from the intersection graph. The zones have been filled with the elements they contain.

7 Conclusions

We presented an heuristic for the construction of intersection graphs as described in [9]. The algorithm presented attempts to optimise the rules that characterise a good intersection graph, obtaining satisfactory results in terms of the quality of the output.

This algorithm represents the first step of a completely automatic method for the generation of Euler representations. We are currently developing a method where the constructed intersection graph is drawn along with the class boundaries, in order to obtain drawings similar to the one manually generated in figure 8(b).

References

- [1] Florence Benoy and Peter Rodgers. Evaluating the comprehension of euler diagrams. In *IV*, pages 771–780. IEEE Computer Society, 2007.
- [2] Stirling Christopher Chow. *Generating and drawing area-proportional Euler and Venn diagrams*. PhD thesis, 2007.
- [3] Leonhard Euler. Lettres à une princesse d’allemagne, letters no. 102-108, 1761.
- [4] Andrew Fish and Gem Stapleton. Defining euler diagrams: choices and consequences. *Euler Diagrams 2005*.
- [5] Andrew Fish and Gem Stapleton. Formal issues in languages based on closed curves. In *Distributed Multimedia Systems*, pages 161–167, 2006.
- [6] Jean Flower and John Howse. Generating euler diagrams. *Lecture Notes in Computer Science*, 2317, 2002.
- [7] Peter Rodgers, Leishi Zhang, and Andrew Fish. General euler diagram generation. volume 5223, pages 13–27. Springer, September 2008.
- [8] Peter Rodgers, Leishi Zhang, Gem Stapleton, and Andrew Fish. Embedding wellformed euler diagrams. *12th International Conference on Information Visualisation*, 2008.
- [9] Paolo Simonetto and David Auber. Visualise undrawable euler diagrams. In *IV08*. IEEE Computer Society, July 2008.
- [10] Gem Stapleton, Peter Rodgers, John Howse, and John Taylor. Properties of euler diagrams. *Electronic Communications of the EASST*.
- [11] Anne Verroust and Marie-Luce Viaud. Ensuring the drawability of extended euler diagrams for up to 8 sets. In *Diagrammatic Representation and Inference*, volume 2980 of *Lecture Notes in Computer Science*, pages 128–141. Springer, 2004.