DOI: 10.1111/cgf.14814

Eurographics Conference on Visualization (EuroVis) 2023
D. Archambault, R. Bujack, and T. Schreck
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 42 (2023), Number 3

RectEuler: Visualizing Intersecting Sets using Rectangles

Patrick Paetzold[1], Rebecca Kehlbeck[1], Hendrik Strobelt[2], Yumeng Xue[1],
Sabine Storandt[1], and Oliver Deussen[1]

[1]University of Konstanz, Germany [2]IBM Research AI, US

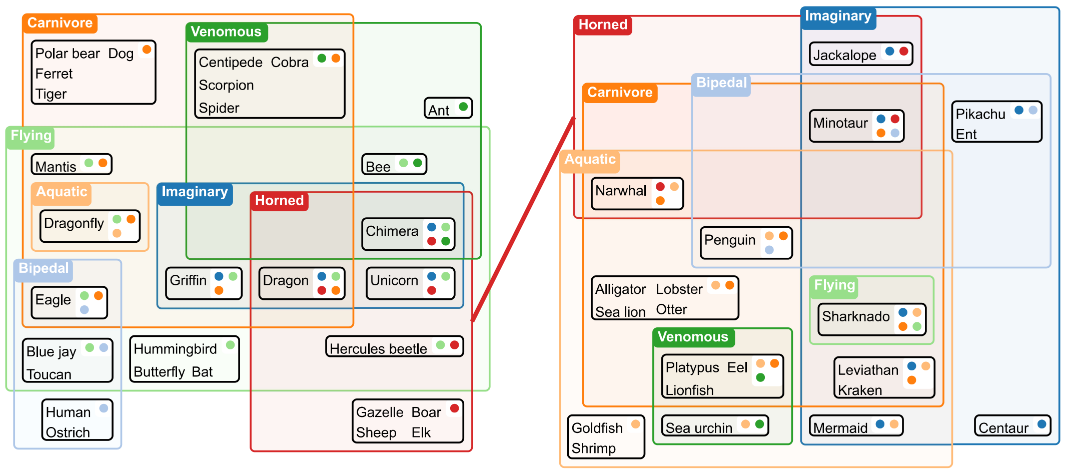


Figure 1: *Animals and their attributes visualized by RectEuler: The optimization yields compact rectangles for each set. For complex datasets with many highly-overlapping regions it is not always possible to represent each set using a single rectangle. In these cases, the visualization splits into connected sub-layouts, which is the case for the 'Horned' set. Our generation method is flexible and enables user customization of several properties, such as set colors, sizes, and contents.*

Abstract

Euler diagrams are a popular technique to visualize set-typed data. However, creating diagrams using simple shapes remains a challenging problem for many complex, real-life datasets. To solve this, we propose RectEuler: a flexible, fully-automatic method using rectangles to create Euler-like diagrams. We use an efficient mixed-integer optimization scheme to place set labels and element representatives (e.g., text or images) in conjunction with rectangles describing the sets. By defining appropriate constraints, we adhere to well-formedness properties and aesthetic considerations. If a dataset cannot be created within a reasonable time or at all, we iteratively split the diagram into multiple components until a drawable solution is found. Redundant encoding of the set membership using dots and set lines improves the readability of the diagram. Our web tool lets users see how the layout changes throughout the optimization process and provides interactive explanations. For evaluation, we perform quantitative and qualitative analysis across different datasets and compare our method to state-of-the-art Euler diagram generation methods.

CCS Concepts

• ***Human-centered computing*** $\rightarrow$ ***Visualization; Visualization techniques;***

1. Introduction

Joint relations across data elements are a common occurrence for many datasets. Examples exist in various fields such as bioinformatics [DDA*12, JXW21], infographics for science [Wil12], and in popular media [Hag13, MAS19]. Different approaches and metaphors have been introduced to generate visualizations that show such set relations in a readable and space-efficient way. Many existing techniques represent sets as shapes enclosed by two-dimensional curves and intersections of sets by overlaying intersecting regions. These diagrams are called *Euler diagrams*. An obvious advantage of Euler visualization techniques over other set visualizations is their ability to incorporate elements directly into the diagram. This helps users understand the data, as they can read individual data elements and quickly identify the sets they are contained in. Previous work aimed at creating Euler diagrams using different shapes, e.g., rectangles, circles [RSA*16], ellipses [KGWD21], and splines [WRD21], as seen in Fig. 2. While these approaches can visualize complex datasets, they often have drawbacks in other areas, e.g., misrepresenting set relation structures, creating difficult-to-read line intersections, no ability to incorporate data elements into the visualization automatically, or applicability to only a few datasets.

Therefore, it is still challenging to create readable Euler diagrams that automatically incorporate individual elements while improving on these issues simultaneously. Our proposed work aims to resolve these problems, creating diagrams where the sets' elements are an integral part of the visualization while the layout remains compact and guarantees that all intersections present in the set relation structure are included. Adopting a multiple-component metaphor with different splitting strategies of the set intersection makes it possible to draw any dataset - even highly intersecting Venn diagrams. This extension enables the creation of very comprehensible visualizations, as shown in the teaser.

We use an efficient mixed-integer optimization with carefully selected constraints to create Euler-like diagrams for any set-typed data. We support user interactions in our tool, allowing users to get insights into the data utilizing well-established interaction techniques such as hovering, zooming, and panning. In addition, we directly integrate explanations about the quality of areas of the diagrams. We show the usefulness and characteristics of our method on multiple examples from different domains, such as infographics, entertainment, and nutrition. A comparison to several state-of-the-art techniques, such as EulerView, MetroSets, and Edeap demonstrates the aesthetic advantages of our method. In summary, the main contributions of this paper are:

- A label-agnostic, fully-automatic method for generating customizable, compact rectangular Euler diagrams tailored to set-typed data.
- An interactive tool that supports the user's sense-making process and gives insights into the quality of the diagrams.
- Evaluation of multiple real-world datasets that demonstrate our approach's wide range of applicability a direct comparison to state-of-the-art methods.

2. Related Work

We focus on the automatic generation of Euler diagrams. Thus, other well-established set visualization techniques like UpSet [LGS*14], KelpFusion [MRS*13] or BubbleSets [CPC09] are not discussed. For additional information, we refer to the extensive overview provided by Alsallakh et al. [AMA*14] for set visualizations and Rodgers [Rod14] for Euler diagrams.

Euler-like Diagrams with integrated Data Elements

For these approaches, data elements are automatically integrated into the diagram, e.g., as labels. Simonetto and Auber [SA08] introduce desired properties of an Euler diagram and the corresponding implications on the intersection graph. As a follow-up [SA09], they propose a concrete implementation using heuristics to insert edges into an initially empty intersection graph. An algorithm (PrEd) to generate an Euler diagram given an intersection graph is given in [SAA09]. Elements of the sets are inserted into the regions, and PrEd iteratively refines the layout. In addition to colors, they use textures to make the overlapping sets more distinctive. EulerView [Sim11] is available as an extension to the information visualization framework tulip [tul].

Riche and Dwyer propose ComED [RD10a] and DupED [RD10a], two approaches using rectangles as the geometric primitive to generate Euler-like diagrams. In contrast to most other Euler-like diagrams, they do not enclose all elements of a set in a single shape but use links between multiple rectangles to indicate an element's set membership. Elements that are part of multiple sets are duplicated, and instances are added to all appropriate rectangles. Both EulerView and ComED/DupED can integrate data elements directly. EulerView visualizations, however, get unreadable quickly as set outlines lie directly on top of each other. In contrast, ComED/DupED uses simpler shapes, but as elements are duplicated, this will result in cluttered visualizations for highly intersecting sets. Further, the lines created by ComED intersect and many different angles, hindering readability.

A more general approach to visualize relations using rectangular elements was proposed by Yoghourdjian et al. [YDG*16]. They use a linear programming (LP) approach to visualize graph relations on a grid. Their approach is also transferable to the problem of creating rectangular Euler diagrams and offers an important initial solution. However, its applicability to drawing Euler diagrams is limited — the method only works well on small datasets that do not have complex intersections. For many real-world datasets, no diagram can be created. Furthermore, the layout is limited to fixed-sized cells and focused on visualizing networks.

More recently, MosaicSets [RWB*22] revisited the idea of solving Euler-like diagrams using LPs, extending previous work using rectangular grids to more complex layouts, such as hexagonal cells. Additional complexity is introduced, as each grid cell is modeled individually, but the goal is to find a continuous region so that all cells remain connected. On top, they can overlay another set relation of the same data elements, where curves enclose these cells to form a continuous area. They show that the mapping to such a 2D layout is NP-hard. Compared to our method, they focus primarily on diagrams with few sets and no highly overlapping regions. Their line intersections are more regular, similar to ComED, but the closeness of lines might hinder readability.

Area-proportional Euler diagrams

Area-proportional Euler diagrams focus on generating layouts with areas proportional to a predefined value, e.g., the number of elements in this area or another domain-specific value.

Perez-Silva et al. [PSAVQ18] propose nVenn, a method to draw area-proportional Euler diagrams based on a combination of graph decomposition and a force-based layout. First, a Venn diagram with all possible regions is generated by applying the approach described in [RSW06], which uses a symmetric chain decomposition. It ensures that a curve can enclose each region of a Venn diagram. A force-based optimization technique is used to optimize the diagram and shrink the regions. Curves are replaced by springs, contracting the zones and thus compacting the layout. As the curves are optimized to enclose the circles tightly, concurrent curves are created.

Rodgers et al. introduce in their overview publication [RSA*16] the layout technique SetNet for Euler diagrams using circles. It relies on the iCircles algorithm [SFRH12]. iCircles generates an initial layout by splitting regions of set intersections not representable by one circle. Thus, more than one circle can represent a set in the Euler diagram. However, faces may become arbitrarily small or large, hindering readability. Afterward, a hill-climbing approach is used to move the circles to simplify the placement of nodes inside the circles representing the set elements.

Wybrow et al. [WRD21] introduce a visualization system for area-proportional Euler diagrams using circles and ellipses. They provide a web-based implementation of their system [ede]. A hill-climbing algorithm is used to minimize an objective function, that is, among other things, based on the difference between the current area and the desired area of zones. Edeap provides numerical feedback about the error between each region's actual and desired area. However, since the final Euler diagram does not necessarily faithfully represent the sets and their intersections present in the dataset, it might introduce empty regions or omit regions in the diagram. Therefore, Edeap is not well-matched.

Other approaches

Here, we will discuss approaches that do not fit into the previous two categories but are still relevant to our approach.

A new way of merging two already existing and popular visualization techniques—adjacency matrices and node-link diagrams—was proposed in NodeTrix [HFM07]. The authors of NodeTrix created a hybrid representation of large graphs and their relations. Combining these techniques allows reading both global and local structures in a unified visualization. Their key contributions are a collection of interaction techniques and smooth transitions from one state to the other.

In contrast to the previously introduced methods using curves and their enclosed areas to represent sets and their relation, Jacobsen et al. [JWKN21] use the well-established style of octolinear metro maps to convey the relation of sets and their elements. Intersections of sets, in Euler diagrams represented by two-dimensional regions, are instead represented by collateral metro lines. Metro lines are often drawn on an octolinear grid to increase their readability and reduce visual clutter [WJKN21]. An example of the metro metaphor can be seen in Fig. 8a.

Kehlbeck et al. [KGWD21] recently introduced a graph-based technique to generate Euler diagrams with curved splines by creating a planar layout of the dual of the Euler diagram. The final Euler visualization can be created by arranging this planar graph in a circular form and cutting the edges of the dual for each set. The method works very well for highly-connected Euler diagrams and Venn diagrams. However, it cannot automatically integrate individual data elements or create area-proportional visualizations in their current form. Another drawback is the current limitation of only being able to draw connected datasets—this is an unrealistic assumption for most real-world datasets.

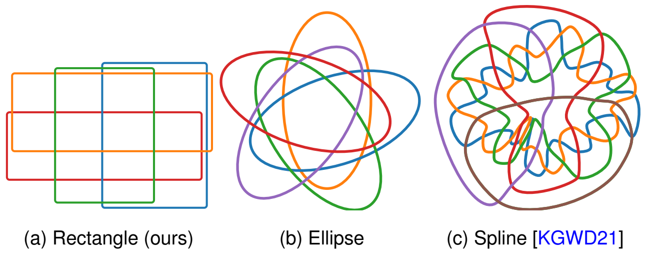


Figure 2: *Axis-aligned rectangles can visualize full sets up to four, ellipses up to five, and more complex curves must be used for any other full sets.*

To visualize hypergraphs, Qu et al. [QZZ22] propose a method based on the layout of regular polygons sharing vertices. It works well for datasets with shallow intersections, and the hypergraph has a tree-like structure. For data where one vertex shares many polygons, unwanted artifacts arise.

In summary, previous methods already proposed initial solutions to create rectangular Euler diagrams. However, they have several drawbacks. For one, their main focus is not on set-typed but on other data types, so they are not optimized for the specific issues of real-world set datasets. Another problem is that their results do not scale well with highly overlapping data, which is often the case for real-world datasets, and introduce hard-to-read line intersections. In our work, we focus on improving the applicability of set-typed data, considering their often highly complex intersections.

3. Desirable properties of Euler diagrams

Euler diagrams use intersecting curves to describe relations between data elements. Each curve represents one set, while the areas enclosed by the curves represent a specific intersection of sets, also called an intersection zone. As depth we describe the number of participating sets of an intersection zone. The abstract description describes all set intersections that exist in a given dataset. Creating a 'good' Euler diagram is challenging because many different properties influence the readability of such a diagram. Kehlbeck et al. [KGWD21] give an overview of different methods and how they conform to desired properties of being well-matched and well-formed, based on the properties suggested by Blake et al. [BSRH16]. Well-matched means that the diagram exactly represents the intersections of the abstract description faithfully, and well-formed describes adhering to several form aspects, e.g., no triple intersections, no concurrency, and simple curves. However, to further improve the adoption of Euler diagrams in real-world applications, we argue that additional considerations should be taken into account when designing a method for their automatic generation. Although there already exist some methods that create rectangular Euler diagrams, we argue that there is still no method that combines these considerations to a satisfying degree. Therefore, in the following, we will detail why it is so difficult to apply these aspects successfully and how we approached them in our proposed method.

Deciding what kind of curve to use as a set representative has a huge influence on the overall design of the visualization. Many approaches that generate Euler diagrams do so by optimizing circles or ellipses. The guidelines proposed by Blake et al. [BSRH16] directly suggest that this is the best option for the shape of a set curve. Another possibility, e.g., used by spEuler [KGWD21], is to use parametric curves, such as Catmull-Rom splines. However, complex curves are not well-suited for large-scale optimizations. An alternative for simple primitives is using rectangular curves. Rectangular diagrams were used by Chow et al. [CR04] and further discussed by Blake et al. [BSR*14], but declared inferior to circles. However, we argue that this statement needs to be discussed more. When comparing manually created Euler diagrams to automatically-generated ones, we saw that many human-made diagrams use rectangles to represent set curves. Although MosaicSets [RWB*22] can create stunning visualizations with compact shapes, for complex datasets, the set lines intersect at different, sometimes small angles, creating a difficult-to-read visualization [HHE08, HEH14]. This brings us to the conclusion that rectangular diagrams should not be disregarded so quickly and supports us in considering rectangles as a well-suitable geometric primitive for Euler diagrams.

A common addition to Euler diagrams is the possibility of directly integrating the visual representation of data elements inside the diagram. The diagrams should be, to some extent, area-proportional – to be able to adapt the shape of individual zones to the size of their data. Many direct optimization techniques that create area-proportional diagrams do so without integrating the data elements themselves but only using their weight to adapt the shape of a zone. However, individual element representations might be of varying shapes that cannot be described by weight. Therefore, it would be better to directly model and integrate their shape during optimization. Furthermore, the visualization should remain compact: MetroSets [JWKN21] is a method that works for almost all datasets and can show each element found in the data. Because their initial optimization does not consider all data elements, the intersection graph is modeled with one element per set intersection. The remaining data elements are later inserted into the graph, but because of the used Metro map metaphor, they form long splines, potentially creating lots of empty space. Our goal was to integrate individual elements from the very beginning. Therefore, they are considered at every step of our optimization. In theory, any data can be directly included in the diagram as long as it can be described using a bounding rectangle. We can also integrate different forms of representation and encode additional attributes in the size of the representative.

The problem of deciding if an intersection system can be drawn with rectangles is related to the notion of boxicity. Here, an intersection system is given as a graph of nodes (sets) and links (sets sharing an intersection). The boxicity of the graph defines if it can be embedded in n dimension using intersecting rectangles, with each rectangle representing one set. Deciding if a graph, in general, has boxicity 2 is NP-hard [Kra94], although some algorithms exist to solve it for specific cases [ACS14] or unreasonable runtime [QW90]. In our case, the problem of drawing an Euler diagram is even stricter. For boxicity, it does not matter how sets are connected to each other. However, given an abstract description, a specific set intersection has to be realized. Our goal is to be able to produce diagrams for any dataset, but we cannot decide beforehand if it can be drawn easily. Therefore, we split the dataset into parts, draw a diagram for each part, and connect them to each other.

To summarize, our method aims to create diagrams with simple curves that can be adaptable and robust to the form of input. The method should always be able to create a visualization, even for challenging datasets with highly overlapping sets.

4. Optimization

In the following, we will demonstrate that generating a rectangular Euler diagram can be formulated as a (Mixed) Integer Linear Programming problem. They can be solved using a standard optimizer, in our case, the Gurobi solver. Our approach is similar to Yoghourdjian et al. [YDG*16]. For more details about its internal structure and the specific algorithms that are used to solve Mixed Integer programs, please refer to its documentation [gur].

4.1. Linear Programming

We use the following definition of linear programming problems proposed by Winston and Goldberg [WG04, p. 53]: "A *linear programming problem* (LP) is an optimization problem for which we do the following:

1. We attempt to minimize (or maximize) a linear function of the decision variables. The function to be maximized or minimized is called the objective function.
2. The values of the decision variables must satisfy a set of constraints. Each constraint must be a linear equation or linear inequality.
3. A sign restriction is associated with each variable. For any variable x_i, the sign restriction specifies that x_i must be either nonnegative ($x_i \geq 0$) or unrestricted in sign."

In mathematical terms, this can be formulated as

$$\text{minimize } c^T x \text{ subject to } Ax \leq b,\ x \geq 0$$

The matrix $A \in \mathbb{R}^{m \times n}$ represents the constraints. $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are constants of the function to be minimized. The vector $x \in \mathbb{R}^n$ is to be optimized.

All solutions to the linear programming problem are denoted as *feasible region* and can be expressed as the set $\{\mathbb{R}^n | Ax \leq b\}$. Graphically each constraint partitions the n-dimensional space of all possible solutions in two by a hyperplane. All solutions that meet this constraint lie on one side of this hyperplane. All hyperplanes together define a convex polyhedron in $\mathbb{R}^n$ (or the empty region). Many algorithms exist to determine the optimal value of the linear objective function.

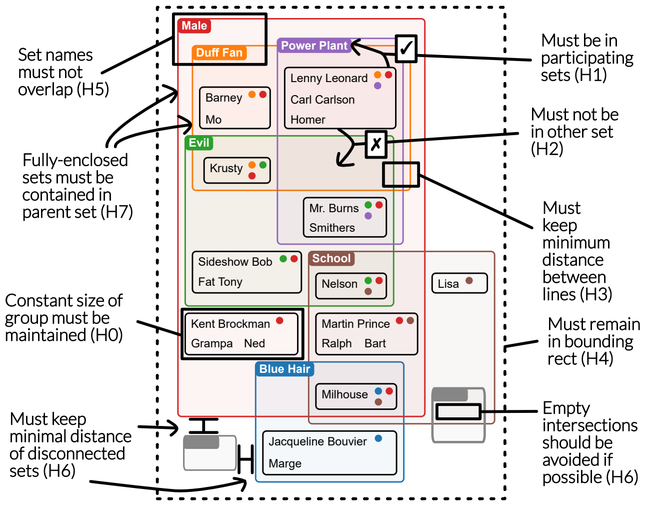

Set names must not overlap (H5)
Fully-enclosed sets must be contained in parent set (H7)
Constant size of group must be maintained (H0)
Must keep minimal distance of disconnected sets (H6)
Must be in participating sets (H1)
Must not be in other set (H2)
Must keep minimum distance between lines (H3)
Must remain in bounding rect (H4)
Empty intersections should be avoided if possible (H6)
Male
Duff Fan
Power Plant
Evil
School
Blue Hair
Lenny Leonard
Carl Carlson
Homer
Barney
Mo
Krusty
Mr. Burns
Smithers
Sideshow Bob
Fat Tony
Nelson
Lisa
Kent Brockman
Grampa
Ned
Martin Prince
Ralph
Bart
Milhouse
Jacqueline Bouvier
Marge

Figure 3: *Overview of the MIP constraints used in our optimization, showcased on the* SIMPSONS *dataset.*

4.2. (Mixed) Integer Linear Programming

Limiting the possible solutions to integer values is necessary for some linear problems. An LP where all variables are required to be integer is called an integer linear program (ILP). If only some variables are required to be an integer, the problem is called a mixed-integer program (MIP). In contrast to linear programs, integer linear programs are NP-hard [CCZ14, p. 20]. Despite their theoretical complexity, even fairly complex MIPs are, in some instances, solvable in a reasonable time by highly optimized state-of-the-art solvers like Gurobi or CPLEX. To solve a *pure integer linear program* or *mixed integer linear program*, the integrality constraints are removed in the first step. The model is thus *relaxed* to a *linear program*. This LP can be solved efficiently and provides a lower bound for minimization or, respectively, upper bound for maximization problems. Moreover, specialized algorithms can use the relaxation and those bounds to solve LIPs like Branch-and-Bound and Cutting Plane [CCZ14].

5. Method

RectEuler uses an optimization that can be applied to any dataset with element-set relations, where one element can be attributed to multiple sets. The data is typically given in form of a table. To allow the user freedom in expressing data elements, labels can be represented with different types, which could be simple text labels or more complex image representatives.

First, we enclose each visual representation of a data element with a bounding rectangle. These rectangles are grouped by their respective set intersections and form an 'element group'. Each element group is again represented with a rectangular shape. We jointly optimize the position and size of the set and element group rectangles using the constraints shown in Fig. 3 using Mixed Integer Programming. Our optimization only produces valid Euler diagrams in the sense that it includes all set intersections defined in the abstract description. Due to the computational complexity, finding an initial solution might take a considerable amount of time. If no arrangement can be found within 50 seconds, we switch to a backup strategy, where the dataset is iteratively split into components until a feasible solution is found. Otherwise, the optimization continues until a cut-off time of 60 seconds. Finally, the user is presented with the visualization that is the basis for further user interactions and visual analysis.

5.1. Constraints

Due to space limitations, we will only provide a brief overview of the constraints used, which can be seen in Fig. 3. Precise definitions can be found in the Supplementary. Similar to the methods proposed by [YDG*16] and [RWB*22], it is essential to define suitable constraints for the LP. Our approach is similar to [YDG*16], as we also constrain the position of elements via their inclusion and exclusion. However, both previous work add further constraints that limit positions on a grid. To create high-quality visualizations while addressing concerns specific to Euler diagrams, we add further constraints. In addition to inclusion and exclusion, we integrate the name of each set directly into the optimization (H5). To reduce the complexity, we minimize the number of constraints, e.g., by checking if a set is completely contained in another one (H7). As intersections without data elements (empty intersections) hinder readability, we add constraints to avoid them if possible (H6).

5.2. Optimization Objective

Constraints H0 to H7 define the relative positions of the rectangles representing the sets and element groups to each other. Thus an Euler diagram generated only using these constraints would contain all necessary set intersections but would not be compact. Each rectangle could become arbitrarily large. Therefore, an objective function is used to formulate an optimization goal to increase the diagram's compactness and draw the rectangles as large as needed.

Minimize Sum of Rectangles The area of each *rectangle r* representing a set should be minimized. This increases the compactness of the rectangular layout. To enforce a tight layout of the individual sets, the half perimeter of all rectangles in the *Universe U* is minimized. The linear objective function is defined as

$$sum_{rect} = \sum_{r \in U} (r_{x_2} - r_{x_1}) + (r_{y_2} - r_{y_1}) \tag{1}$$

Minimize Bounding Rectangle As Eq. 1 only minimizes the size of each individual set, the positions of the sets are not influenced. To generate overall tight layouts, all rectangles are placed in a *bounding rectangle br* by H4. The half perimeter of the bounding rectangle is minimized by

$$sum_{br} = (br_{x_2} - br_{x_1}) + (br_{y_2} - br_{y_1}) \tag{2}$$

Squareness In some cases, diagrams were visually not pleasing because their aspect ratio was large. To counteract this, we add a term to the objective function that reduces the absolute difference between the bounding rectangles' width and height. We added the constraints

$$\begin{aligned} (br_{x_2} - br_{y_2}) &\leq square \\ -(br_{x_2} - br_{y_2}) &\leq square \end{aligned} \tag{3}$$

We combine all sums to the final objective function, which is used to optimize the layout:

$$\text{minimize } sum_{rect} + sum_{br} + square \tag{4}$$

5.3. Optimization Considerations

Initially, we also experimented with adding constraints lazily during optimization to keep the number of constraints low. Adding constraints on conditions is a feature that some optimizers, such as Gurobi, offer. We let the optimization run for small datasets until it finds a globally optimal solution. As the final result will have the same objective value comparing lazy vs. non-lazy constraints, we can directly compare their runtime. In these cases, using lazy constraints sometimes significantly improves the runtime while producing fewer intermediate layouts. However, this is not necessarily the best choice for complex datasets, as it sometimes takes longer to reach the same objective value compared to the non-lazy approach.

In our experience, avoiding undesirable layouts from the beginning resulted in the best solutions, while the runtime differences achieved with lazy constraints were inconclusive. We think this might be partly due to how lazy constraints are handled inside Gurobi, as a lot of the internal processing has to be adapted for them to work. As our primary goal is to produce a visualization reliably, we opted for non-lazy constraints as the default optimization strategy for both simple and complex datasets. In addition, not all MIP solvers offer lazy constraints. By formulating all constraints in advance, our model can be solved by a broader range MIP solvers.

5.4. Splitting Strategies

As not all set-typed data can be represented using one single Euler diagram, as discussed in 3. To overcome this issue, datasets are instead split into multiple sub-layouts. To decide if a dataset can be visualized with just one diagram, we search for an initial solution for the MIP with a time quota of 50 seconds. If a feasible solution is found, the optimization is continued up to a user-defined maximal execution time limit (default 60s), and a diagram and all intermediate solutions are drawn. The 10 seconds are added so that there is a chance to find another layout that is not the initial solution. If no feasible solution is found within the time quota, the optimization is interrupted. We split the dataset instead. We investigated different splitting strategies and their impact on the drawn diagrams. All strategies are iterative, so if a single execution of the strategy does not lead to a solution, we apply it until a diagram can be drawn.

Split randomly We randomly split up the element groups into two disjoint parts. For each part, a MIP is generated and optimized. If one of the parts can be solved, we keep it and continue on with the remaining element groups. The procedure is repeated until all components are representable as individual Euler diagrams. This simple strategy results in somewhat even distribution of elements across the diagrams, but they can still be hard to read as unfavorable combinations of element groups can be created.

Split by clustering To improve upon this, we split in a way that considers the similarity between element groups. We use a binary vector for each element group where a '1' represents the inclusion in a set, and a '0' represents the exclusion. To combine similar element groups, we apply a k-medoids clustering with the Jaccard metric. We start by separating the data into two clusters. If one of the splits can be drawn, we keep it and iteratively cluster the remaining data elements until a feasible solution is found that includes all the data. Incorporating clustering into the splitting strategy keeps similar elements in the same diagram. Our intuition is that this keeps diagrams readable and primarily simple because elements stay connected to other elements that only differ in a few sets, which helps to keep the number of empty intersections down. This is confirmed in Table A1 in the Supplementary, where we can observe that the other splitting strategy creates significantly more and larger empty intersections.

For the cluster strategy, it is sometimes possible that the initial split results in a very uneven distribution of nodes across the diagrams. In that case, we try to find another split three times and otherwise accept the last split.

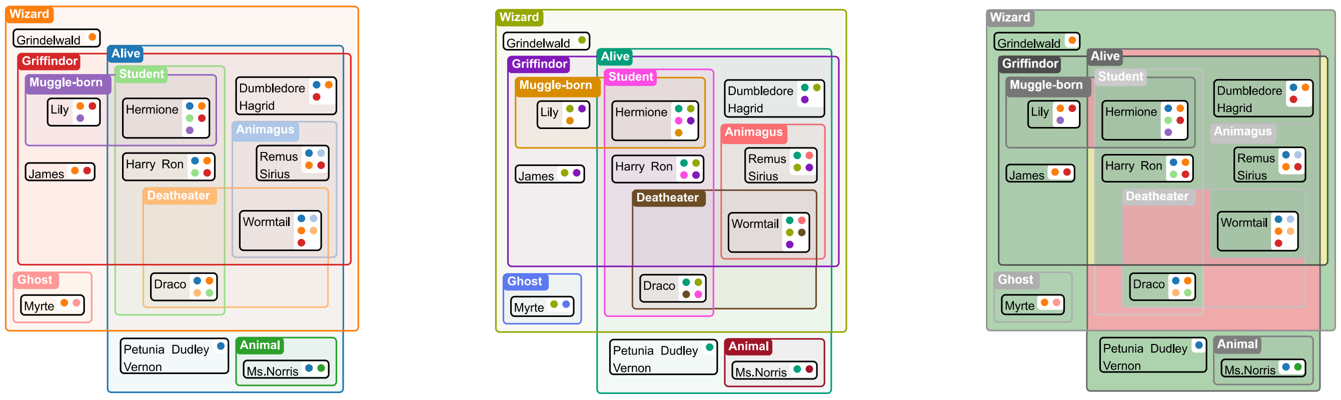
5.5. Visual Design Considerations

Not only where elements are placed in the diagram, but other visual factors, such as set color or additional attributes, influence the readability and usability of the visualization.

Redundant Encoding As noted in [BSR*14] and [BSRH14], common tasks for Euler diagrams are to identify in which and in how many sets an element is contained. For complex abstract descriptions, the resulting Euler diagrams are challenging to understand. Many sets interact and form highly overlapping regions. To support the user in fulfilling tasks, *redundant coding* may be used. It is, e.g., frequently used in scatter plots to distinguish points not only by color but additionally by shape [War19]. As it is not immediately evident to the user to which sets an element group belongs, we add small colored dots inside each element group, inspired by Upset [LGS*14]. Thus, the set inclusion of an element is expressed by its position and the colored dots next to it. It is sufficient to count the number of dots to quickly identify the number of sets an element group is contained in. As this redundant coding is not based on user interaction, it can also be used in static Euler diagrams.

Color We offer a range of different possibilities to assign colors to set rectangles. As a default, the qualitative color pallets introduced by Tableau are used to provide distinctive colors (Fig. 4a). However, randomly assigning colors to the rectangles means that rectangles that are close to each other might get assigned similar colors. To remedy this, we adapted the color assignment proposed by Pallettailor [LFC*21]. We sample the outline of each set and assign dissimilar colors to adjacent rectangles. In Fig. 4a, 'Animagus' and 'Alive' have similar colors, which is not the case using Palettailor, as seen in Fig. 4b. Still, the user might be dissatisfied with the automatically selected colors. To customize the diagram further, he can interact with the label on each rectangle and select his preferred color in a live preview color wheel picker.

Connecting Lines To maintain a visual connection between the split diagrams, we use connecting lines between the rectangles representing a set, visually similar to the connections presented



(a) Tableau20 (b) Pallettailor (c) Highlighted area types

Figure 4: *HARRY POTTER dataset: The user can choose between the Tableu20 (a) and Pallettailor (b) color scheme and investigate the quality of the diagram by highlighting duplicated (yellow) and empty intersections (red) (c).*

in [RD10b]. To reduce visual clutter, we show the connections only when hovering over the names of the sets. An example of this can be observed in Fig. 1, where the rectangles of the set 'Horned' in red are connected.

Encoding additional data attributes The labels representing each data element can be of different types, depending on the user's choice. They could be simple text labels, images, or other shapes. Additionally, it is possible to encode data attributes on the size of each label. An example of the EU dataset encoding each country's GDP as the size of the flag is Fig.A10 in the Supplementary.

6. RectEuler Tool

We implemented a preliminary version of our method with a python back-end and a JavaScript front-end (`https://rectvis.de/`)

6.1. In-place Interactions and Explanations

Besides creating the diagrams and using them as static visualizations, our interface also offers insights into the properties of the diagram and its set intersection system. We offer several statistics to help the user get an overview of the dataset and help him analyze the quality of the produced diagrams. These include information about the abstract description, the runtime, the number of duplicated sets, and the number and area percentage of the empty intersections and their distribution across different depths.

Interactions By encoding the set membership using the colored dots on the right side of each element group, it is immediately apparent in which and in how many sets an element is contained. The user can hover over an element group; all sets the element group belongs to are highlighted. Set names are attached to each set rectangle, making a separate legend unnecessary. This makes the visualization completely self-contained, as all set information is encoded inside the diagram. Hovering on a set label reduces the opacity of all other sets and connects the set across the diagrams, should there be a split. The tool provides a timeline to follow the optimization process. Each correct layout solution can be individually selected and investigated. Fig. 7 shows an example of this evolution. In case the user is not satisfied with the automatic color selection, it is possible to change them manually. By clicking on a set label, a color wheel opens and the color can be freely selected. The change is automatically updated in the diagram to make the selection easier.

Intersection Area Coloring On-demand, we color the areas of the diagram according to their existence in the abstract description: empty intersections that do not contain any elements (red), empty duplicates of another area (yellow), and areas that have a 1-to-1 equivalent in the abstract description (green). The user can enable an overlay that colors all areas according to their type and quickly identify unwanted areas (Fig. 4c).

7. Results

We quantitatively analyze the impact of different splitting strategies and the occurrence of empty intersections as a measure of the quality of the diagrams. To illustrate the effects of the optimization, we also show intermediate solutions of the MIP in Fig. 7. Finally, we conclude this chapter with a summary of adherence concerning well-formedness and well-matchedness properties.

To provide a broad overview of the method's strengths and weaknesses, we use datasets with various numbers of elements, set cardinalities, and overlap. For scalability, we tested datasets with up to 20 sets and 360 elements. A mixture of already established and new datasets were chosen for evaluation. We use the SIMPSONS and ANIMALS datasets provided by Upset 2 [GSGL19] , as well as ROCK'N'ROLL, and MARVEL datasets by MetroSets [met]. An overview of the datasets with their characteristics, runtime, sub-layouts, and empty intersection characteristics can be found in Table 1. All results are generated on a desktop PC with an Intel i7-7700K and 32GB of RAM. We used Gurobi version 9.5.2 and its Python 3.8 bindings. We can observe that the runtime increases with the number of set intersections. However, there are also outliers where this does not seem to be the main factor (e.g., Marvel or

Table 1: *Overview of dataset properties, runtimes, and empty intersections statistics. * signifies datasets where an optimal solution was found. First results for datasets with multiple sub-layouts include the time used for infeasible solutions.*

datasets	#sets	#elements	#zones	first result (s)	first objective	final result (s)	final objective	#sub layouts	#duplicated sets	%area empty	#empty intersections	time used for infeasible solutions (s)
Venn 4*	4	15	15	0.1	2768	1.1	2744	1	0	0	0	0
Simpsons*	6	21	11	0.1	4976	12.2	4185	1	0	8.9	4	0
Vitamins*	6	31	13	0.1	5083	21.6	4243	1	0	12.1	6	0
Fruit*	9	10	9	0.1	8066	28.7	6692	1	0	73.3	33	0
Rock'n'Roll	8	225	21	0.2	13432	21.9	10689	1	0	4.4	5	0
Magic	12	112	27	0.4	19436	38.6	12977	1	0	13.7	13	0
EU	17	50	26	1.2	21989	243.0	17431	1	0	54.5	33	0
Euler-like Vis. Papers	20	53	24	20.1	16131	298.5	13589	1	0	11.3	32	0
Names	8	25	15	0.1	6574	52.3	5038	1	0	17.2	16	0
Harry Potter*	9	17	12	0.1	7293	13.8	5295	1	0	17.1	7	0
Mosaic Fac. 2*	8	51	14	0.1	11152	5.4	7602	1	0	11.4	5	0
DnD	8	359	68	145.2	51806.0	233.2	40943.0	4	8	14.6	36.0	143.9
IMDb	10	90	41	91.9	27965.0	130.7	21995.5	3	9	20.7	20.0	91.0
Animals	7	50	29	50.3	12183.5	104.3	9951.0	2	7	15.1	10.0	50.0
Marvel	28	24	24	93.3	40711.0	140.4	36050.0	2	13	78.6	113.0	50.3
Venn 5	5	31	31	13.8	9333.5	16.7	8357.5	3	5	11	7.5	13.6
Mosaic Nat. 8	13	178	33	52.2	19258.0	97.0	15333.5	2	6.5	13.1	15.0	50.3

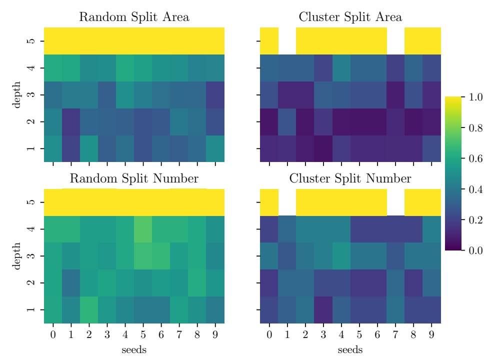


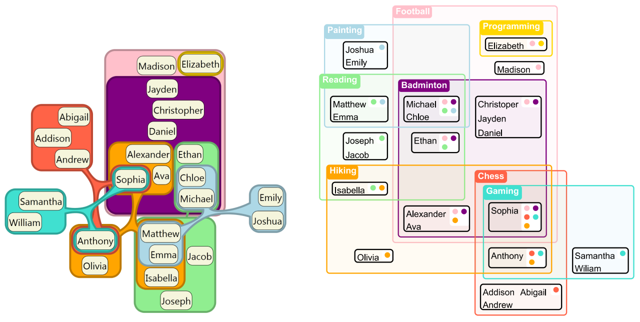
Figure 5: *Heatmap of differently seeded splits for the ANIMALS dataset and the percentage and distribution of empty intersections.*

EU). Therefore, a more thorough investigation is needed. Because the optimization can take a very long time to complete, depending on the dataset, it is not always viable to wait for the optimal result. For most of the datasets in Table 1, it was set to 60s. Only for the datasets EU and Euler-Like Vis. Papers we optimize for 300s. These quotas were chosen as they might be a reasonable time that a user would wait for a result. They are similar to the 300s cut-off used by [YDG*16]. However, this quota could be extended by the user if waiting time and computation queues are not an issue. For split datasets, we used the median value of ten different seeded versions, as the splits are not deterministic. We provide additional examples in the supplementary material.

7.1. Analysis of Empty Intersections

The quality of an Euler diagram can be measured using different approaches, from the number of intersections to its aspect ratio. MetroSets proposed multiple quantitative measures for their approach. However, most of them do not apply to diagrams and are concerned with the visual quality of the method. Another possibility to measure the quality is the area error. But this information does not give us any insight into why the error was introduced and the complexity of the data. Therefore, we propose analyzing the area and count of unwanted, empty intersections for a dataset. We investigated datasets according to their rank distribution and subsequent creation of unwanted areas to get more insights into the relationship between unwanted zones and their corresponding abstract descriptions. For split datasets, we averaged across several random initialization seeds. We plot the number of overlapping sets (depth of intersection) as a heatmap in Fig. 5. Cells in the heatmap are colored according to the ratio of either the number or area of undesirable intersections in comparison to the total number or area of intersections of a given depth. An intersection is undesirable if it is either not at all present in the abstract description or a duplicated zone (red and yellow regions in Fig. 4c). Overall, the random split introduced more empty intersections across all depths. The percentage of empty intersections per seed stays relatively uniform for well-connected datasets, and it is improbable to get stuck with an unfavorable split. Although it is not directly linked to the drawability of the diagram, it may provide valuable information for analyzing the data. Our intuition is that the undrawable diagrams often occur because of conflicting intersections on nodes of the same rank - the K5 intersection graph is one example of this. Therefore, if a diagram is not drawable using a single diagram, it might be interesting to see the distribution of unwanted zones for each sub-layout and the overall distribution across all diagrams.

Optimization Evolution In the interactive tool, users can see the evolution of the optimization process based on the objective function. If the user can see how the original layout is organized step-by-step into the final compact layout, this may reinforce his understanding of the original data sets and relationships within the data, similar to the transitions proposed by Henry et al. [HFM07]. Al-



(a) ComED [RD10b] (b) Our method (Colors of ComED)

Figure 6: *PERSON dataset: our approach (right) allows us to show the complex relations without additional connecting segments.*

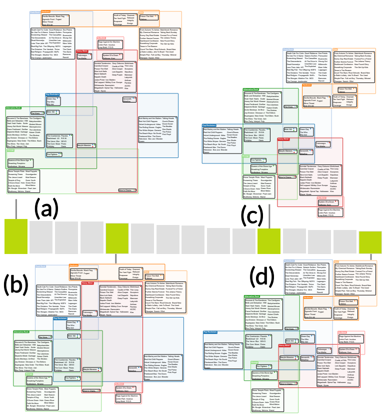


Figure 7: ROCK'N'ROLL *Iterations: (a) 1st, 0.2s (b) 5th, 2.7s (c) 11th, 7.2s (d) 15th, 21.9s. The bars show the decrease in objective value and the tightening of the bounding rectangles.*

though the initial result of our optimization is already correct in the sense that all zones of the abstract description are included, it will not be compact (Fig. 7). The first feasible solution illustrated in Fig. 7(a) was obtained after 0.2 seconds. It could be more visually pleasing since the element groups are far apart; thus, the diagram is dominated by empty space. The subsequent iterations increase the compactness significantly. In the 15th iteration, as illustrated in Fig. 7(d), the diagram's general layout changed extensively, resulting in a very compact diagram.

7.2. Euler Diagram Properties

We will shortly summarize how our work relates to the established guidelines by Blake et al. [BSRH16].

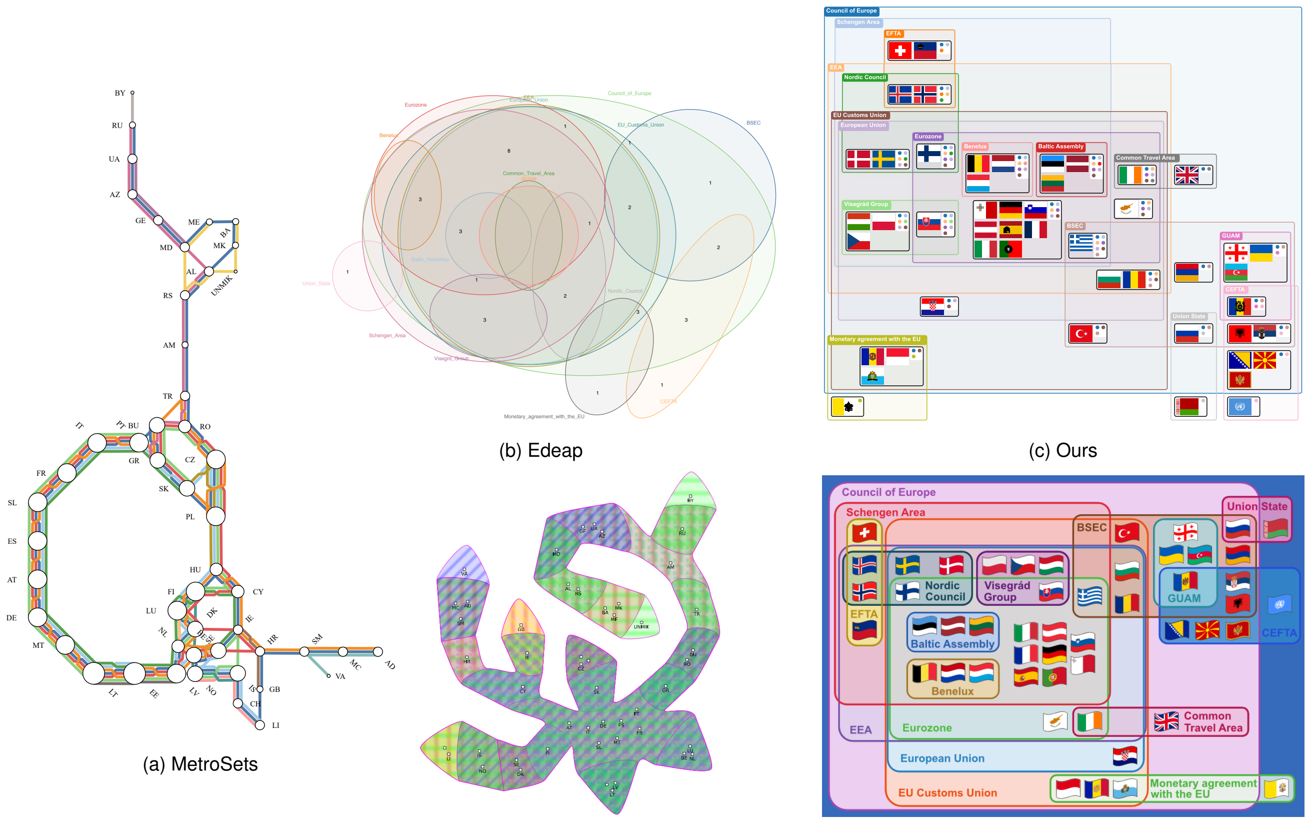
Well-formedness Our method generates Euler diagrams without triple points, brushing points, or concurrent lines. Due to the rectangular shapes in our Euler diagrams, line segments run in parallel, but constraint H3 creates a minimal distance between them. Triple points cannot occur, as lines can only ever cross from two different directions, and for any additional set curve, the previous constraint would enforce the minimal distance. Similarly, brushing points are avoided by enforcing a minimal margin between two non-intersecting rectangles. Thus the concurrency criterion is met, as no horizontal nor vertical line segment lays on top of another line segment. As rectangles cannot self-intersect, all curves in our method are simple. If we do not split the data, each set is represented by exactly one rectangle. Otherwise, some set curves will be duplicated. Our method does not prevent disconnected zones. Even for simple datasets, disconnects may occur. We highlight these unwanted areas, as shown in Fig. 4c in yellow.

Well-matchedness Both methods we introduced in Section 2 that produce well-matched Euler diagrams (EulerView [Sim11], spEuler [KGWD21]) use splines as outlines. Like all other Euler diagrams using simple geometric shapes such as circles or ellipses, our method cannot generate well-matched results. It may include empty intersections that are not present in the abstract description or duplicate intersections. However, compared to related work, we can guarantee that all intersections of the abstract description exist in the diagram. Unwanted areas can be highlighted on demand using the interactive tool (red intersection areas in Fig. 4c).

8. Comparison to Related Work

Using a complex example, we will discuss how our method compares to available related work and afterward outline limitations and future work. Due to space constraints, we will only discuss the EU dataset (see Fig. 8). A large-scale version of the EU comparison, as well as results for other datasets and comparison to more related works, are found in the supplementary material. This dataset is based on the hand-crafted visualization shown in Fig. 8e. It describes the membership of countries to supranational institutions. We chose this dataset because it is available as a heavily tweaked hand-drawn Euler diagram. It is challenging as it contains many sets fully enclosed by other sets and elements that are part of many different sets. For some previous methods [RD10a, YDG*16], we could not compare ourselves directly, as no implementation is available, or it cannot be used to generate Euler diagrams.

Fig. 8 shows how we compare to the original [8e], human-made visualization, as well as MetroSets [8a], Edeap [8b] and EulerView [8d]. Other methods, such as SetNet or spEuler, did not produce a diagram for this dataset. As we can immediately tell, Edeap has problems correctly visualizing the individual set intersections and some intersections are missing. It creates a lot of unwanted zones, as well as concurrent segments. EulerView creates a well-matched result. However, it comes at the cost of maximizing concurrent segments. These make it hard to read. Because the data contains many highly overlapping sets, MetroSets also shows many concurrent segments. However, because each set intersection also contains many data points, the final result takes up considerable space. Our result seems not too far from the human-crafted results. However, to get insights regarding the difference and preferences of human-crafted vs. automatically created visualizations, a larger user study across several quantitative measures is needed. It allows users to specify the shape and type of each element, in this case, an image of a flag. The diagram in Fig. 8c contains unwanted, empty set intersections, but for complex datasets such as this one, it is almost unavoidable. However, we can produce a compact layout with



(a) MetroSets (b) Edeap (c) Ours (d) EulerView (e) Human-made (Figure from WikiMedia [Wdc21])

Figure 8: *EU dataset, visualization of super-national Institutions of the EU.*

no direct concurrency. Although it is not as compact or regular as the original visualization, it gets reasonably close.

8.1. Limitations and Future Work

A current limitation is the placement of sub-layouts. We place them horizontally and connect duplicated sets of neighboring diagrams. For future work, it might be interesting to investigate a more flexible placement strategy. To improve runtime and scalability, it would be possible to exempt disconnected sets from the optimization and place them using a heuristic approach. Improving the splits by incorporating them in the MIP optimization could be an interesting aspect for further research.

Compared to the human-made Euler diagram of the EU dataset shown in Fig. 8e, our diagram is less dense. The fixed placement of the set's name in the rectangle's upper left corner and the element groups make our layout less flexible. Individual elements cannot float freely in comparison to the human-made example. This drawback could be alleviated by extending the method with a force-based layout, in which items such as labels or element groups can move freely inside their intersection zone.

9. Conclusion

We have presented a fully-automatic visualization method for generating compact two-dimensional rectangular Euler-like diagrams. Our method generates a single chart if all elements and set rectangles can be drawn in a single Euler diagram. Otherwise, the dataset is split into multiple sub-layouts. Our layout technique can display elements of different sizes and types as an integral part of the visualization. We use a *Mixed Integer Program* to model the layout of the sets and their elements and define several MIP constraints to describe the geometric relation of sets to each other and the elements. Its generated diagrams can be used as both static and interactive visualizations. Static, since all information about the sets, such as their labels and the respective elements, are already part of the visualization and interactive, as the user interface provides several interactions and explanations to help the users read and understand the Euler diagram. Our method is fully-automatic and can therefore be used in scenarios where the user does not want to invest any manual effort to layout a diagram. It creates Euler diagrams in seconds up to minutes. As we combine multiple diagrams in a connected visualization, any dataset can be visualized. In the future, we would like to implement that users can move individual elements and change the edges of the set curves. In general, more interactivity might be helpful.

Acknowledgments

This work was funded by the German Research Foundation (DFG) Project-ID 251654672 – TRR 161 "Quantitative methods for visual computing" as well as KE 740/17-2 of the FOR2111 "Questions at the Interfaces". Open Access funding enabled and organized by Projekt DEAL.

References

[ACS14] Adiga A., Chandran L. S., Sivadasan N.: Lower bounds for boxicity. *Combinatorica 34*, 6 (jun 2014), 631–655. doi:10.1007/s00493-011-2981-0. 4

[AMA*14] Alsallakh B., Micallef L., Aigner W., Hauser H., Miksch S., Rodgers P.: Visualizing sets and set-typed data: State-of-the-art and future challenges. The Eurographics Association. doi:10.2312/EUROVISSTAR.20141170. 2

[BSR*14] Blake A., Stapleton G., Rodgers P., Cheek L., Howse J.: The impact of shape on the perception of Euler diagrams. In *8th Int. Conf. on Diagrammatic Representation and Inference* (2014), vol. 8578, pp. 123–137. doi:10.1007/978-3-662-44043-8_16. 4, 6

[BSRH14] Blake A., Stapleton G., Rodgers P., Howse J.: How should we use colour in euler diagrams? In *Proc. of the 7th Int. Symposium on Visual Information Communication and Interaction* (2014), Association for Computing Machinery, p. 149–158. doi:10.1145/2636240.2636838. 6

[BSRH16] Blake A., Stapleton G., Rodgers P., Howse J.: The impact of topological and graphical choices on the perception of Euler diagrams. *Information Sciences 330* (2016), 455–482. doi:10.1016/j.ins.2015.05.020. 3, 4, 9

[CCZ14] Conforti M., Cornuéjols G., Zambelli G.: *Integer Programming*. Springer International Publishing, Cham, 2014. doi:10.1007/978-3-319-11008-0_9. 5

[CPC09] Collins C., Penn G., Carpendale S.: Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. & Comp. Graphics 15*, 6 (2009), 1009–1016. doi:10.1109/TVCG.2009.122. 2

[CR04] Chow S., Ruskey F.: Drawing area-proportional venn and euler diagrams. In *Graph Drawing*. Springer Berlin Heidelberg, 2004, pp. 466–477. doi:10.1007/978-3-540-24595-7_44. 4

[DDA*12] D'Hont A., Denoeud F., Aury J.-M., Baurens F.-C., Carreel F., Garsmeur O., Noel B., Bocs S., Droc G., Rouard M., et al.: The banana (musa acuminata) genome and the evolution of monocotyledonous plants. *Nature 488*, 7410 (2012), 213–217. doi:10.1038/nature11241. 1

[ede] Edeap. https://www.eulerdiagrams.org/edeap/. Accessed: 2022-11-02. 3

[GSGL19] Gadhave K., Strobelt H., Gehlenborg N., Lex A.: Upset 2: From prototype to tool. In *Proceedings of the IEEE Information Visualization Conference – Posters (InfoVis '19)* (2019). 7

[gur] Gurobi. https://www.gurobi.com/. Accessed: 2022-11-02. 4

[Hag13] Hagy J.: *How to Be Interesting: (In 10 Simple Steps)*. Workman Publishing Company, 2013. 1

[HEH14] Huang W., Eades P., Hong S.-H.: Larger crossing angles make graphs easier to read. *Journal of Visual Languages and Computing 25*, 4 (aug 2014), 452–465. doi:10.1016/j.jvlc.2014.03.001. 4

[HFM07] Henry N., Fekete J.-D., McGuffin M. J.: NodeTrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (nov 2007), 1302–1309. doi:10.1109/tvcg.2007.70582. 3, 8

[HHE08] Huang W., Hong S.-H., Eades P.: Effects of crossing angles. In *2008 IEEE Pacific Visualization Symposium* (mar 2008), IEEE. doi:10.1109/pacificvis.2008.4475457. 4

[JWKN21] Jacobsen B., Wallinger M., Kobourov S., Nöllenburg M.: Metrosets: Visualizing sets as metro maps. *IEEE Trans. Vis. & Comp. Graphics 27*, 2 (2021), 1257–1267. doi:10.1109/TVCG.2020.3030475. 3, 4

[JXW21] Jia A., Xu L., Wang Y.: Venn diagrams in bioinformatics. *Briefings Bioinform. 22*, 5 (2021). doi:10.1093/bib/bbab108. 1

[KGWD21] Kehlbeck R., Görtler J., Wang Y., Deussen O.: speuler: Semantics-preserving euler diagrams. *IEEE Trans. Vis. & Comp. Graphics* (2021), 1–1. doi:10.1109/TVCG.2021.3114834. 2, 3, 4, 9

[Kra94] Kratochvíl J.: A special planar satisfiability problem and a consequence of its NP-completeness. *Discrete Applied Mathematics 52*, 3 (aug 1994), 233–252. doi:10.1016/0166-218x(94)90143-0. 4

[LFC*21] Lu K., Feng M., Chen X., Sedlmair M., Deussen O., Lischinski D., Cheng Z., Wang Y.: Palettailor: Discriminable colorization for categorical data. *IEEE Transactions on Visualization and Computer Graphics 27*, 2 (feb 2021), 475–484. doi:10.1109/tvcg.2020.3030406. 6

[LGS*14] Lex A., Gehlenborg N., Strobelt H., Vuillemot R., Pfister H.: Upset: Visualization of intersecting sets. *IEEE Trans. Vis. Comput. Graph. 20*, 12 (2014), 1983–1992. doi:10.1109/TVCG.2014.2346248. 2, 6

[MAS19] Maggie Astor D. L., Stevens M.: Who's in the democratic debates, and who's in danger of missing them, 2019. URL: https://www.nytimes.com/interactive/2019/04/29/us/politics/democratic-primary-debates-2020.html. 1

[met] Metroset dataset. www.osf.io/dnpws/. Accessed: 2022-11-02. 7

[MRS*13] Meulemans W., Riche N. H., Speckmann B., Alper B., Dwyer T.: Kelpfusion: A hybrid set visualization technique. *IEEE Trans. Vis. Comput. Graph. 19*, 11 (2013), 1846–1858. doi:10.1109/TVCG.2013.76. 2

[PSAVQ18] Pérez-Silva J. G., Araujo-Voces M., Quesada V.: nVenn: generalized, quasi-proportional Venn and Euler diagrams. *Bioinformatics 34*, 13 (02 2018), 2322–2324. doi:10.1093/bioinformatics/bty109. 2

[QW90] Quest M., Wegner G.: Characterization of the graphs with boxicity $\leqslant 2$. *Discrete Mathematics 81*, 2 (apr 1990), 187–192. doi:10.1016/0012-365x(90)90151-7. 4

[QZZ22] Qu B., Zhang E., Zhang Y.: Automatic polygon layout for primal-dual visualization of hypergraphs. *IEEE Trans. Vis. Comput. Graph. 28*, 1 (2022), 633–642. URL: https://doi.org/10.1109/TVCG.2021.3114759, doi:10.1109/TVCG.2021.3114759. 3

[RD10a] Riche N. H., Dwyer T.: Untangling Euler diagrams. *IEEE Trans. Vis. & Comp. Graphics 16*, 6 (2010), 1090–1099. doi:10.1109/TVCG.2010.210. 2, 9

[RD10b] Riche N. H., Dwyer T.: Untangling euler diagrams. In *IEEE Trans. Vis. & Comp. Graphics* (3 2010), IEEE, pp. 1090–1099. 7, 9

[Rod14] Rodgers P.: A survey of euler diagrams. *Journal of Visual Languages & Computing 25*, 3 (2014), 134–155. doi:https://doi.org/10.1016/j.jvlc.2013.08.006. 2

[RSA*16] Rodgers P., Stapleton G., Alsallakh B., Michallef L., Baker R., Thompson S.: A task-based evaluation of combined set and network visualization. *Information Sciences 367-368* (2016), 58–79. doi:https://doi.org/10.1016/j.ins.2016.05.045. 2, 3

[RSW06] Ruskey F., Savage C. D., Wagon S.: The search for simple symmetric venn diagrams. *Notices Amer. Math. Soc 53* (2006), 1304–1312. 3

[RWB*22] ROTTMANN P., WALLINGER M., BONERATH A., GEDICKE S., NOLLENBURG M., HAUNERT J.-H.: MosaicSets: Embedding set systems into grid graphs. *IEEE Transactions on Visualization and Computer Graphics* (2022), 1–11. doi:10.1109/tvcg.2022.3209485. 2, 4, 5

[SA08] SIMONETTO P., AUBER D.: Visualise undrawable euler diagrams. In *2008 12th International Conference Information Visualisation* (2008), pp. 594–599. doi:10.1109/IV.2008.78. 2

[SA09] SIMONETTO P., AUBER D.: An heuristic for the construction of intersection graphs. In *2009 13th International Conference Information Visualisation* (2009), pp. 673–678. doi:10.1109/IV.2009.30. 2

[SAA09] SIMONETTO P., AUBER D., ARCHAMBAULT D.: Fully automatic visualisation of overlapping sets. *Computer Graphics Forum 28*, 3 (2009), 967–974. doi:https://doi.org/10.1111/j.1467-8659.2009.01452.x. 2

[SFRH12] STAPLETON G., FLOWER J., RODGERS P., HOWSE J.: Automatically drawing euler diagrams with circles. *Journal of Visual Languages & Computing 23*, 3 (2012), 163–193. doi:https://doi.org/10.1016/j.jvlc.2012.02.001. 3

[Sim11] SIMONETTO P.: *Visualisation of Overlapping Sets and Clusters with Euler Diagrams. (Diagrammes d'Euler pour la visualisation de communautés et d'ensembles chevauchants)*. PhD thesis, University of Bordeaux, France, 2011. 2, 9

[tul] Tulip. https://tulip.labri.fr/. Accessed: 2022-11-02. 2

[War19] WARE C.: *Information visualization: perception for design*. Morgan Kaufmann, 2019. 6

[Wdc21] WDCF AND THE EMIRR AND NIKNAKS93: Supranational european bodies-en.svg, 2021. [Online; accessed 11-February-2022]. URL: https://upload.wikimedia.org/wikipedia/commons/archive/6/6a/20220310193917%21Supranational_European_Bodies.svg. 10

[WG04] WINSTON W. L., GOLDBERG J. B.: *Operations Research: Applications and algorithms*. Thomson Brooks/Cole Belmont, 2004. 4

[Wil12] WILKINSON L.: Exact and approximate area-proportional circular Venn and Euler diagrams. *TVCG 18*, 2 (2012), 321–331. doi:10.1109/tvcg.2011.56. 1

[WJKN21] WALLINGER M., JACOBSEN B., KOBOUROV S., NOLLENBURG M.: On the readability of abstract set visualizations. *IEEE Trans. Vis. & Comp. Graphics 27*, 6 (2021), 2821–2832. doi:10.1109/tvcg.2021.3074615. 3

[WRD21] WYBROW M., RODGERS P., DIB F.: Euler diagrams drawn with ellipses area-proportionally (edeap). *BMC Bioinformatics 22* (04 2021). doi:10.1186/s12859-021-04121-8. 2, 3

[YDG*16] YOGHOURDJIAN V., DWYER T., GANGE G., KIEFFER S., KLEIN K., MARRIOTT K.: High-quality ultra-compact grid layout of grouped networks. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (2016), 339–348. doi:10.1109/TVCG.2015.2467251. 2, 4, 5, 8, 9