LUND
UNIVERSITY

Bachelor Thesis

eulerr: Area-Proportional Euler Diagrams with Ellipses

Johan Larsson

supervised by
Peter Gustafsson

January 1, 2018

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Science in Statistics
at the Department of Statistics, Lund University

Contents

Abstract

Euler diagrams are common and intuitive visualizations for data involving sets and relationships thereof. Compared to Venn diagrams, Euler diagrams do not require all set relationships to be present and may therefore be area-proportional also with subset or disjoint relationships in the input. Most Euler diagrams use circles, but circles do not always support accurate diagrams. A promising alternative for Euler diagrams is ellipses, which, in theory,enable accurate diagrams for a wider range of input. Ellipses, however, have not yet been implemented for more than three sets or three-set diagrams where there are disjoint or subset relationships. The aim of this thesis is to present a method and software for elliptical Euler diagrams for any number of sets.

In this thesis, we provide and outline an R-based implementation called eulerr. It fits Euler diagrams using numerical optimization and exact-area algorithms through two steps: first, an initial layout is formed using the sets' pairwise relationships; second, this layout is finalized taking all the sets' intersections into account.

Finally, we compare eulerr with other software implementations of Euler diagrams and show that the package is overall both more consistent and accurate as well as faster for up to seven sets compared to the other R-packages. eulerr perfectly reproduces samples of circular Euler diagrams as well as three-set diagrams with ellipses, but performs suboptimally with elliptical diagrams of more than three sets. eulerr also outperforms the other software tested in this thesis in fitting Euler diagrams to set configurations that might lack exact solutions provided that we use ellipses; eulerr's circular diagrams, meanwhile, fit better on all accounts save for the diagError metric in the case of three-set diagrams.

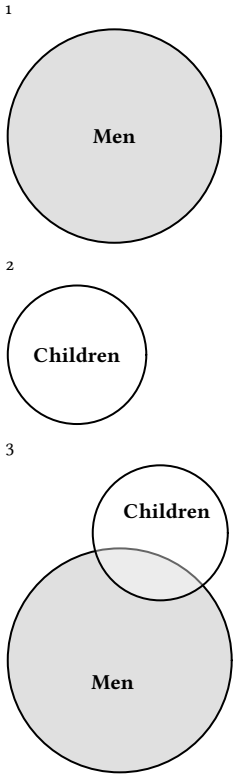
Keywords: *Euler diagrams, Venn diagrams, ellipses, R, computer graphics, area-proportional, software*

1 Introduction

1.1 Background

Visual displays of data make for clear and compelling presentations, utilizing multiple dimensions to inform concisely and intuitively. Compared to tables and text, visualization possess the potential to convey even intricate relationships with effective use of ink.

Data visualizations, however, are only effective when they illustrate relationships. Consider, for instance, a disc labelled *Men*[1]—it says nothing by itself; yet if we juxtapose it with another, smaller, disc labelled *Children*[2], the graphic starts to become informative, now displaying the relative sizes of two sets. If next we intersect the two discs, producing an overlap[3], we successfully visualize the relative proportions of men and children, as well as their intersection. The diagram we have constructed is an *Euler diagram*.



The Euler diagram, originally proposed by Leonard Euler in 1802 [1], is a generalization of the obiquiteous *Venn diagram*: a staple of introductory text books in statistics and research disciplines such as biomedicine and geology. Venn and Euler diagrams both display set relationships by mapping an area of the diagram to a relationship in the data. They differ, however, in that the Venn diagrams require all intersections to be present—even if they are empty—which Euler diagrams do not.

Euler diagrams may moreover be area-proportional, which is to say that each separate surface of the diagram maps proportionally to its data. (This is the case with the diagram to the right.) This is a rational form for a Euler diagram—only its geometry is needed to interpret it. And it lets us, for instance, to discard numbers without losing critical information; the same cannot be said for a Venn diagram.

Area-proportional Euler diagrams may be fashioned out of any closed shape, and have been implemented for triangles [2], rectangles [2], ellipses [3], smooth curves [4], polygons [2], and circles [2, 5, 6]. The latter are most common, and for good reason, since they are easiest to interpret [7]. Circles, unfortunately, do not always support accurate diagrams. Consider the following three-set configuration:

$$\begin{gathered} A = B = C = 2, \\ A \cap B = A \cap C = B \cap C = 1, \text{ and} \\ A \cap B \cap C = 0. \end{gathered}$$

There is no way to visualize this relationship perfectly with circles because they cannot be arranged to keep the $A \cap B \cap C$ overlap empty whilst $A \cap B$, $A \cap C$, and $B \cap C$ remain non-empty. With ellipses, however, we can solve this problem since they can both rotate and stretch, enabling a perfect fit (Figure 1.1).

With four or more sets that all intersect, exact Euler diagrams are in fact impossible with circles, given that we require 15 intersections but with four circles can yield at most 13 unique overlaps. This is not the case with ellipses, which may intersect in up to four, rather than two, points, altogether yielding the necessary 15 unique areas. As of yet, the only implementation of elliptical Euler diagrams is provided in **eulerAPE** [3], although it only supports three sets that are all required to intersect. The diagram in Figure 1.2, for instance, would be impossible with **eulerAPE**.

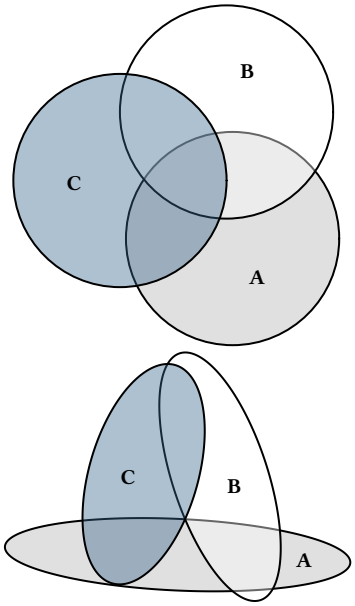


Figure 1.1. A set relationship depicted erroneously with circles and perfectly with ellipses.

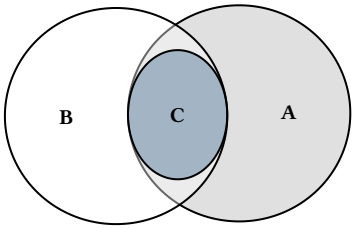


Figure 1.2. An Euler diagram with a subset relationship.

Euler diagrams have to be solved numerically [8]. Most software accomplish this in two steps, first finding a coarse starting layout that is finalized in a second, more thorough, algorithm. For the initial layout, the aforementioned **eulerAPE** package [9], for instance, uses a greedy algorithm that tries to minimize the error in the three-way intersection by arranging the circles representing the sets. The **venneuler** package [5], meanwhile, uses multidimensional scaling (MDS), taking only pairwise relationships into account. **venn.js** [10] combines a *constrained* version of the MDS algorithm from **venneuler** with a greedy algorithm[4], picking the best fit out of the two. **Vennerable** [2] computes the required pairwise distances between circles and then adjusts the largest of these to optimize the two-way overlaps of the layout. All of the above use circles in the initial layout.

Diagrams with more than two sets normally require additional tuning, for which we must first find the areas of the overlaps in the diagram, so that we can scrutinize our diagram's fit. Computing these overlaps, however, is no trivial task—particularly not for ellipses. For this reason, many methods resort to approximations such as quadtrees [5], which are used in **venneuler**, or polygon intersections [6].

Compared to approximative algorithms, exact algorithms require that we know the intersection points of the ellipses. There are two known approaches to this. Both treat the ellipses in pairs. The first method [11] solves the system of equations formed by each pair of ellipses, which involves solving a fourth-degree polynomial; the other method [12] represents the ellipses as conics in projective geometry, which reduces to solving a third-degree polynomial. Both methods are accurate up to floating-point precision.

With all the intersection points at hand, it is possible to derive the areas of the overlaps. Frederickson [10] (**venn.js**) and Micallef and Rodgers [3] (**eulerAPE**) have developed solutions for circles and ellipses respectively—although the latter, as we previously covered, restricts itself to three intersecting ellipses. No algorithm has

[4] This greedy algorithm places the sets sequentially in order of size.

so far been published that extends these methods to any number of ellipses or elliptical three-set diagrams with subset or disjoint intersections.

In the final layout, the package's area-algorithm is used with a numerical optimizer to tune the parameters of the diagram. All previously considered packages treat this as a minimization problem but their loss functions vary. **venn.js**, for instance, uses the residual sums of squares[5], **venneuler** uses the *stress* metric[6], **Vennerable** uses Chow's [13] *idealistic function*[7], and **eulerAPE** uses a proportional loss function[8] that severely punishes missing overlaps—in fact, it becomes undefined if such areas exist, making it inappropriate for algorithms that aim to fit set configurations with either subset or disjoint relationships.

venn.js relies on a *Nelder–Mead optimizer* for the final step. **venneuler**, on the other hand, sports a combination of steepest descent optimization and coordinate descent. **eulerAPE** and **Vennerable**, meanwhile, uses hill climbing algorithms.

1.2 Aims

In this thesis, we aim to present an algorithm and software implementation for constructing and visualizing Euler diagrams for set relationships of any size using ellipses.

We will also compare this method to existing software for Euler diagrams on account of consistency in reproducing diagrams with known, exact solutions, accuracy in finding solutions for set configurations that may lack exact solutions, and computational performance.

[5] **venn.js**'s loss function is

$$\sum_{i=1}^{n} (A_i - \omega_i)^2 ,$$

where n is the number of overlaps, ω_i the size of the i^{th} intersection and its relative complement, and A_i the corresponding area in the diagram.

[6] **venneuler**'s stress metric is defined as

$$\frac{\sum_{i=1}^{n} (A_i - \beta\omega_i)^2}{\sum_{i=1}^{n} A_i^2},$$

where $\beta = \sum_{i=1}^{n} A_i\omega_i / \sum_{i=1}^{n} \omega_i^2$.

[7] This idealistic function (used in **Vennerable**) is defined as

$$\sum_{i=1}^{n} \left(\frac{\omega_i}{\sum_{i=1}^{n} \omega_i} - \frac{A_i}{\sum_{i=1}^{n} A_i} \right)^2 + \sum_{i<j<n} \left(A_i - A_j\right),$$

where the sets and their respective overlaps corresponding to the indices i and j have been ordered so that $i < j \implies \omega_i < \omega_j$.

[8] **eulerAPE**'s cost function is defined as

$$\frac{1}{n} \sum_{i=1}^{n} \frac{(\omega_i - A_i)^2}{A_i}.$$

2 Method

Constructing an Euler diagram is much like fitting a statistical model in that we have

1. data,
2. a model to fit the data with,
3. tests to assess the model's fit, and
4. a presentation of the result.

In the following sections, we explain how **eulerr** tackles each item in turn.

2.1 Input

Euler diagrams present relationships between sets, wherefore the data must describe these relationships, either directly or indirectly. **eulerr** allows several alternatives for this data, namely,

- intersections and relative complements[9],
- unions and identities[10],
- a matrix of binary (or boolean) indices[11],
- a list of sample spaces[12], or
- a two- or three-way table[13].

As an additional feature for the matrix form, the user may supply a factor variable with which to split the data set before fitting the diagram, which sometimes improves diagrams where the set relationships vary across categories.

Whichever type of input is provided, **eulerr** will translate it to the first and second types, *intersections and relative complements* and *unions and identities*, which will be used in the steps to come.

The Euler diagram is then fit in two steps: first, an initial layout is formed with circles using only the sets' pairwise relationships. Second, this layout is fine-tuned taking all $2^N - 1$ intersections into consideration.

[9] $A \setminus B = 3 \quad B \setminus A = 2 \quad A \cap B = 1$

[10] $A = 4 \quad B = 3 \quad A \cap B = 1$

[11]
$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

[12] $A = \{a, b, c, d\}$
$B = \{a, e, f\}$

[13]

	A	A^c
B	1	2
B^c	3	0

2.2 Initial layout

For our initial layout, we adopt a constrained version of multi-dimensional scaling (MDS) that has been adapted from **venn.js** [10], which in turn is a modification of an algorithm used in **venneuler** [5]. In it, we consider only the pairwise intersections between sets, attempting to position their respective shapes so as to minimize the difference between the separation between their centers required to obtain an optimal overlap and the actual overlap of the shapes in the diagram.

This problem is unfortunately intractable for ellipses, being that there is an infinite number of ways by which we can position two ellipses to obtain a given overlap. Thus, we restrict ourselves to circles in our initial layout, for which we can use the circle–circle overlap formula (2.1) to numerically find the required distance, d, for each pairwise relationship.

$$O_{ij} = r_i^2 \arccos\left(\frac{d_{ij}^2 + r_i^2 - r_j^2}{2d_{ij}r_i}\right) + r_j^2 \arccos\left(\frac{d_{ij}^2 + r_j^2 - r_i^2}{2d_{ij}r_j}\right) - \frac{1}{2}\sqrt{(-d_{ij} + r_i + r_j)(d_{ij} + r_i - r_j)(d_{ij} - r_i + r_j)(d_{ij} + r_i + r_j)}, \quad (2.1)$$

where r_i and r_j are the radii of the circles representing the i^{th} and j^{th} sets respectively, O_{ij} their overlap, and d_{ij} their separation.

Setting $r_i = \sqrt{F_i/\pi}$, where F_i is the size of the i^{th} set, we are able to obtain d numerically using the squared difference between O and the desired overlap as loss function (2.2),

$$\mathcal{L}(d_{ij}) = \left(O_{ij} - (F_i \cap F_j)\right)^2, \quad \text{for } i < j \leq n, \quad (2.2)$$

which we optimize using `optimize()`[14] from **stats**.

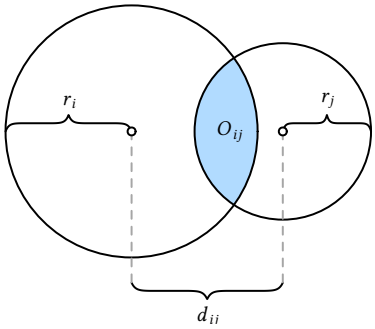


Figure 2.1. The circle–circle overlap is computed as a function of the discs' separation (d_{ij}), radii (r_i, r_j), and area of overlap (O_{ij}).

For a two-set combination, this is all we need to plot an exact diagram, given that we now have the two circles' radii and separation and may place the circles arbitrarily as long as their separation, d, remains the same. This is not, however, the case with more than two sets.

With three or more sets, the circles need to be arranged so that they interfere minimally with one another. In some cases, the set configuration allows this to be accomplished flawlessly, but often, compromises must me made. As is often the case in this thesis, this turns out to be another optimization problem. It can be tackled in many ways; **eulerr**'s approach is based on a method developed by Frederickson [14], which the author describes as constrained multi-dimensional scaling.

The algorithm tries to position the circles so that the separation between each pair of circles matches the separation required from (2.2).

[14] According to the documentation, `optimize()` consists of a "combination of golden section search and successive parabolic interpolation."

If the two sets are disjoint, however, the algorithm is indifferent to the relative locations of those circles as long as they do not intersect. The equivalent applies to subset sets: as long as the circle representing the smaller set remains within the larger circle, their locations are free to vary. In all other cases, the loss function (2.3) is the residual sums of squares of the optimal separation of circles, d, that we found in (2.1), and the actual distance in the layout we are currently exploring.

$$\mathcal{L}(h,k) = \sum_{1 \le i < j \le N} \begin{cases} 0 & F_i \cap F_j = \emptyset \text{ and } O_{ij} = 0 \\ 0 & (F_i \subseteq F_j \text{ or } F_i \supseteq F_j) \text{ and } O_{ij} = 0 \\ \left(\left(h_i - h_j\right)^2 + \left(k_i - k_j\right)^2 - d_{ij}^2 \right)^2 & \text{otherwise} \end{cases}. \tag{2.3}$$

The analytical gradient (2.4) is retrieved as usual by taking the derivative of the loss function,

$$\vec{\nabla} f(h_i) = \sum_{j=1}^{N} \begin{cases} \vec{0} & F_i \cap F_j = \emptyset \text{ and } O_{ij} = 0 \\ \vec{0} & (F_i \subseteq F_j \text{ or } F_i \supseteq F_j) \text{ and } O_{ij} = 0 \\ 4\left(h_i - h_j\right) \left(\left(h_i - h_j\right)^2 + \left(k_i - k_j\right)^2 - d_{ij}^2 \right) & \text{otherwise,} \end{cases} \tag{2.4}$$

where $\vec{\nabla} f(k_i)$ is found as in (2.4) with h_i swapped for k_i (and vice versa).

The Hessian (2.5) for our loss function is given next. However, because the current release of R suffers from a bug[15] causing the analytical Hessian to be updated improperly, the current release of **eulerr** instead relies on the numerical approximation of the Hessian offered by the optimizer.

[15] The current development version of R features a fix for this bug; **eulerr** will be updated to use (2.5) as soon as it is introduced in a stable version of R.

$$\boldsymbol{H}(h_i, k_i) = \sum_{1 \le i < j \le N} \begin{bmatrix} 4\left(\left(h_i - h_j\right)^2 + \left(k_i - k_j\right)^2 - d_{ij}^2\right) + 8\left(h_i - h_j\right)^2 & \cdots & 8\left(h_i - h_j\right)\left(k_i - k_j\right) \\ \vdots & \ddots & \vdots \\ 8\left(k_i - k_j\right)\left(h_i - h_j\right) & \cdots & 4\left(\left(h_i - h_j\right)^2 + \left(k_i - k_j\right)^2 - d_{ij}^2\right) + 8\left(k_i - k_j\right)^2 \end{bmatrix}. \tag{2.5}$$

Note that the constraints given in (2.3) and (2.4) still apply to each element of (2.5) and have been omitted for convenience only.

We optimize (2.3) using the nonlinear optimizer `nlm()` from the R core package **stats**. The underlying code for `nlm()` was written by Schnabel et al. [15]. It was ported to R by Saikat DebRoy and the R Core team [16] from a previous FORTRAN to C translation by Richard H. Jones. `nlm()` consists of a system of Newton-type algorithms and performs well for difficult problems [17].

The initial layout outlined above will sometimes turn up perfect diagrams, but only reliably so when the diagram is completely determined by its pairwise intersections. More pertinently, we have not yet considered the higher-order intersections in our algorithm and neither have we approached the problem of using ellipses—as we set out to do.

2.3 Final layout

We now need to account for all the sets' intersections and, consequently, all the overlaps in the diagram. The goal is to map each area uniquely to a subset of the data from the input and for this purpose we will use the sets' intersections and the relative complements of these intersections, for which we will use the shorthand ω. We introduced this form in Section 2.1, but now define it rigorously in Definition 2.1.

Definition 2.1. *For a family of* N *sets,* $F = F_1, F_2, \ldots, F_N$*, and their* $n = 2^N - 1$ *intersections, we define* ω *as the intersections of these sets and their relative complements, such that*

$$\begin{aligned}
\omega_1 &= F_1 \setminus \bigcap_{i=2}^{N} F_i \\
\omega_2 &= \bigcap_{i=1}^{2} F_i \setminus \bigcap_{i=3}^{N} F_i \\
&\vdots \\
\omega_n &= \bigcap_{i=1}^{N} F_i
\end{aligned}$$

with

$$\sum_{i=1}^{n} \omega_i = \bigcup_{j=1}^{N} F_i.$$

Analogously to ω*, we also introduce the* &*-operator, such that*

$$F_i \& F_j = (F_i \cap F_j) \setminus (F_i \cap F_j)^c.$$

The fitted diagram's area-equivalents for ω will be defined as A, so that an exact diagram requires that $\omega_i = A_i$ for $i = 1, 2, \ldots, 2^N - 1$, where N is the number of sets in the input.

In Section 2.2, we restricted ourselves to circles but now extend ourselves also to ellipses. From now on, we abandon the practice of treating circles separately—they are only a special case of ellipses, and, hence, everything that applies to an ellipse does so equally for a circle.

2.3.1 Intersecting ellipses

As we briefly discussed in Section 1.1, we now need the ellipses' points of intersections. **eulerr**'s approach to this is outlined in Richter-Gebert [12] and based in *projective*, as opposed to *Euclidean*, geometry.

To collect all the intersection points, we naturally need only to consider two ellipses at a time. The canonical form of an ellipse is given by

$$\frac{[(x-h)\cos\phi + (y-k)\sin\phi]^2}{a^2} + \frac{[(x-h)\sin\phi - (y-k)\cos\phi]^2}{b^2} = 1,$$

where ϕ is the counter-clockwise angle from the positive x-axis to the semi-major axis a, b is the semi-minor axis, and h, k are the x- and y-coordinates, respectively, of ellipse's center (Figure 2.2).

However, because an ellipse is a conic[16] it can be represented in quadric form,

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

that in turn can be represented as a matrix,

$$\begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix},$$

which is the form we need to intersect our ellipses. We now proceed to

1. form three degenerate conics from a linear combination of the two ellipses we wish to intersect,
2. split one of these degenerate conics into two lines, and
3. intersect one of the ellipses with these lines, yielding 0 to 4 intersection points points (Figure 2.3).

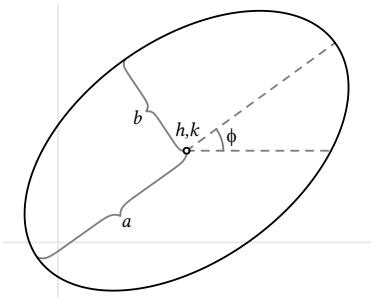
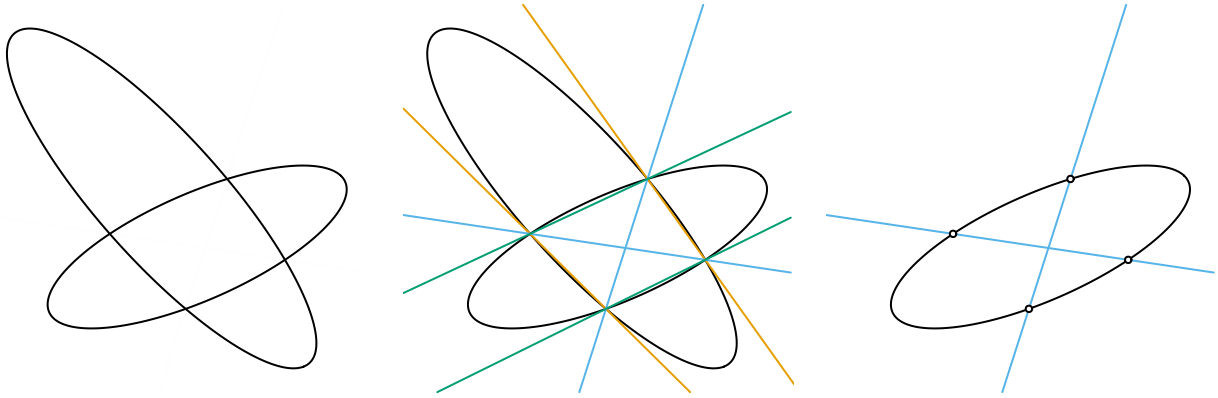


Figure 2.2. A rotated ellipse with semimajor axis a, semiminor axis b, rotation ϕ, and center h, k.

[16] The circle, parabola, and hyperbola are the other types of conics.



(a) Our objective is two intersect these two ellipses.

(b) Three degenerate conics (orange, teal, and blue) are formed from the linear combination of our two ellipses.

(c) One of the degenerate conics is split it into two lines and intersected with one of the ellipses to yield four intersection points.

Figure 2.3. The process used to intersect two ellipses, here yielding four points. This figure was inspired by an example from Richter-Gebert [12].

2.3.2 Overlap areas

Using the intersection points of a set of ellipses that we retrieved in Section 2.3.1, we can now find the overlap of these ellipses. We are only interested in the points that are *contained within all of these ellipses*, which together form a geometric shape consisting of a convex polygon, the sides of which are made up of straight lines between consecutive points, and a set of elliptical arcs—one for each pair of points (Figure 2.4).

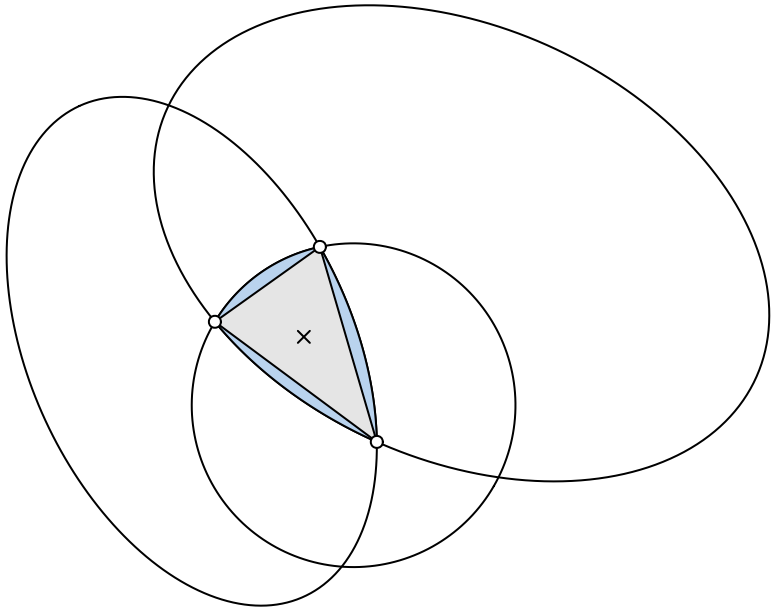


Figure 2.4. The overlap area between three ellipses is the sum of a convex polygon (in grey) and 2–3 ellipse segments (in blue).

We continue by ordering the points around their centroid. It is then trivial to find the area of the polygon section since it is always convex [18]. Now, because each elliptical segment is formed from the arcs that connect successive points, we can establish the segments' areas algorithmically [11]. For each ellipse and its related pair of points (located at angles θ_0 and θ_1 from the semimajor axis), we proceed to find its area by

1. centering the ellipse at $(0, 0)$,
2. normalizing its rotation, which is not needed to compute the area,
3. integrating the ellipse over $[0, \theta_0]$ and $[0, \theta_1]$, producing elliptical sectors $F(\theta_0)$ and $F(\theta_1)$,
4. subtracting the smaller ($F(\theta_0)$) of these sectors from the larger ($F(\theta_0)$, and
5. subtracting the triangle section to finally find the segment area,

$$F(\theta_1) - F(\theta_0) - \frac{1}{2}\left|x_1 y_0 - x_0 y_1\right|,$$

$$\text{where } F(\theta) = \frac{a}{b}\left[\theta - \arctan\left(\frac{(b-a)\sin 2\theta}{b + a + (b-a)\cos 2\theta}\right)\right].$$

This procedure is illustrated in Figure 2.5. Note that there are situations where this algorithm is altered, such that when the sector angle ranges beyond π—we refer the interested reader to Eberly [11].

Finally, the area of the overlap is then obtained by adding the area of the polygon and all the elliptical arcs together.

Note that this does not yet give us the areas that we require, namely A: the area-equivalents to the set intersections and relative complements from Definition 2.1. For this, we must decompose the overlap areas so that each area maps uniquely to a subspace of the set configuration. This, however, is simply a matter of transversing down the hierarchy of overlaps and subtracting the higher-order overlaps from the lower-order ones. For a three-set relationship of sets A, B, and C, for instance, this means subtracting the $A \cap B \cap C$ overlap from the $A \cap B$ one to retrieve the equivalent of $(A \cap B) \setminus C$.

The exact algorithm may in rare instances[17], break down, the culprit being numerical precision issues that occur when ellipses are tangent or completely overlap. In these cases, the algorithm will approximate the area of the involved overlap by

1. spreading points across the ellipses using Vogel's method (see Appendix A.1 for a brief introduction),
2. identifying the points that are inside the intersection via the inequality

$$\frac{[(x-h)\cos\phi + (y-k)\sin\phi]^2}{a^2} + \frac{[(x-h)\sin\phi - (y-k)\cos\phi]^2}{b^2} < 1,$$

   where x and y are the coordinates of the sampled points, and finally
3. approximating the area by multiplying the proportion of points inside the overlap with the area of the ellipse.

With this in place, we are now able to compute the areas of all intersections and their relative complements, ω, up to numerical precision.

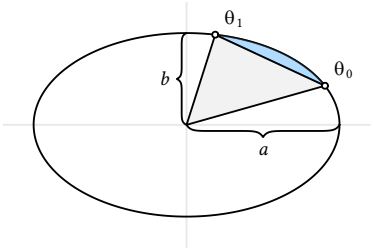


Figure 2.5. The elliptical segment in blue is found by first subtracting the elliptical sector from $(a, 0)$ to θ_0 from the one from $(a, 0)$ to θ_1 and then subtracting the triangle part (in grey).

[17] 1 out of approximately 7000 in our simulations.

2.3.3 Final optimization

We feed the initial layout computed in Section 2.2 to the optimizer—once again we employ `nlm()` from **stats** but now also provide the option to use ellipses rather than circles, allowing the "circles" to rotate and the relation between the semiaxes to vary, altogether rendering five parameters to optimize per set and ellipse (or three if we restrict ourselves to circles). For each iteration of the optimizer, the areas of all intersections are analyzed and a measure of loss

returned. The loss we use is the same as that in **venneuler** [5], namely *stress*,

$$\frac{\sum_{i=1}^{n}(A_i - \beta\omega_i)^2}{\sum_{i=1}^{n} A_i}, \tag{2.6}$$

where

$$\beta = \frac{\sum_{i=1}^{n} A_i\omega_i}{\sum_{i=1}^{n} \omega_i^2}.$$

This is equivalent to linear regression through the origin, where β is the slope of the regression line (Figure 2.6).

2.3.4 Last-ditch optimization

If the fitted diagram is still inexact after the procedure in Section 2.3.3, we offer a final step in which we pass the parameters on to a last-ditch optimizer. The weapon of choice[18] is a *differential evolution algorithm* from the R package **RcppDE** [19]—a port of the **DEoptim** package [20] from C to C++.

The solutions offered by **RcppDE** often avoid local minima but may be inefficient in local search regions; this shortcoming can be remedied by fine tuning with a local optimizer [21]—once more, we rely on `nlm()` to serve this purpose.

By default, this last-ditch step is activated only when we have a three-set diagram with ellipses and a diagError (2.7) above 0.001[19] The reason being that the method is considerably more computationally intensive.

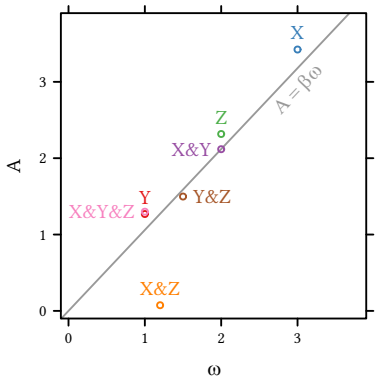


Figure 2.6. Optimizing via stress is analogous to least-squares linear regression through the origin. ω is the set of unique quantities in the input (Definition 2.1) and A the respective areas in the diagram.

[18] We conducted thorough benchmarking, that we opt not to report here, to decide upon an algorithm for this step.

[19] The choice of if and when this last-ditch optimizer is activated is left to the user via simple commands to the main function of the package.

2.4 Goodness of fit

Every Euler diagram must be investigated for its adequacy in representing the input. Exact Euler diagrams are not always possible When **eulerr** cannot find a perfect solution, it offers an approximate one instead, the adequacy of which has to be measured in a standardized way. For this purpose we adopt two measures: *stress* [5], which is also the loss metric we use in our final optimization step and is used in **venneuler**, as well as *diagError* [3], which is used by **eulerAPE**.

We first encountered the stress metric (2.6) in Section 1.1 and covered it again in Section 2.3. The stress metric is not easily grasped but can be transformed into a rough analogue of the correlation coefficient via $r = \sqrt{1 - \text{stress}^2}$.

diagError, meanwhile, is given by

$$\max_{i=1,2,\ldots,n} \left| \frac{\omega_i}{\sum_{i=1}^{n} \omega_i} - \frac{A_i}{\sum_{i=1}^{n} A_i} \right|, \tag{2.7}$$

which is the maximum *absolute* difference of the proportion of any ω to the respective unique area of the diagram.

2.5 Availability

eulerr is available as an R package on the CRAN network [22] and is installed by calling

```
install.packages("eulerr")
```

A development version, along with source code, for the project is maintained in a GitHub repository at https://github.com/jolars/eulerr. This version can be installed, provided that the **devtools** package [23] is installed, by running the following code:

```
devtools::install_github("jolars/eulerr")
```

We have also developed a **shiny** [24] web application for **eulerr**, which can be found at http://jolars.co/eulerr. It features a slimmed-down version of the package that allows the two primary forms of input, offers a little less freedom in customizing the diagram, and does not feature the last-ditch optimizer (Section 2.3.4).

Finally, the source code for this thesis has been provided in a GitHub repository at https://github.com/jolars/eulerr2017bsc, which, in addition to the text and markup for the thesis, also hosts the code used to generate the data, the data itself, and the analyzes of this data (Chapter 3).

3 Results

The only R packages that feature area-proportional Euler diagrams are **eulerr**, **venneuler**, **Vennerable**, and **d3VennR**. The latter is an interface to the **venn.js** script that has been discussed previously, but because it features an outdated version of the script and only produces images as html, we call **venn.js** directly using the javascript engine **V8** via the R package of the same name [25]. Only **eulerr**, **venn.js**, and **venneuler** support more than three sets, which is why there are only three-set results for **Vennerable** and **eulerAPE**.

The packages used here were

- **eulerr** 3.1.0,
- **eulerAPE** 3.0.0,
- **venn.js** 0.2.14,
- **venneuler** 1.1-0, and
- **Vennerable** 3.1.0.9000.

The results for **eulerAPE** were computed on a laptop computer[20]. The remaining results were computed on an Amazon EC2 cloud-based computing instance[21].

3.1 Case studies

We begin our examination of **eulerr** by studying a difficult set relationship from Wilkinson [5],

$$
\begin{gathered}
A = 4, \quad B = 6, \quad C = 3, \quad D = 2, \quad E = 7, \quad F = 3, \\
A\&B = 2, \quad A\&F = 2, \quad B\&C = 2, \quad B\&D = 1, \\
B\&F = 2, \quad C\&D = 1, \quad D\&E = 1, \quad E\&F = 1, \\
A\&B\&F = 1, \quad \text{and} \quad B\&C\&D = 1,
\end{gathered}
$$

where we use the & operator as in Definition 2.1. We fit this specification with **venneuler** and **eulerr**, in the latter case using both circles and ellipses.

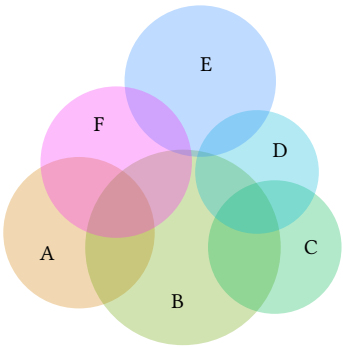
eulerr manages to fit this set configuration perfectly using ellipses (Figure 3.1a) and furthermore produces a better circular Euler diagram (Figure 3.1b) compared to **venneuler** (Figure 3.1a), although the difference is small.

[20] The specification of the computer was

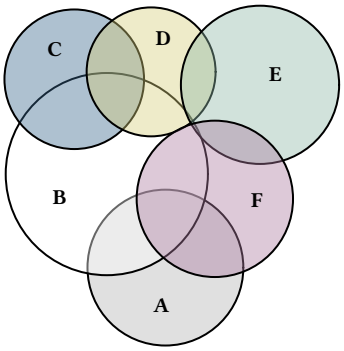
- Microsoft Windows Pro 10 x64
- Intel® Core™ i7-4500U CPU @ 1.80GHz, 2 cores
- 8 GB memory

[21] This was a m4.large instance with the following specifications:

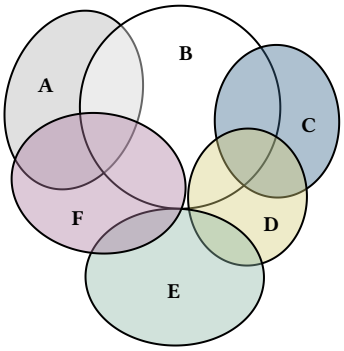
- Ubuntu 16.04 x64
- 2.4 GHz Intel Xeon®E5-2676 v3 (Broadwell) CPU, 2 cores
- 8 GB memory



(a) The diagram from **venneuler**, with a stress of 0.007 and diagError of 0.013.



(b) The circular diagram from **eulerr**, with a stress of 0.004 and diagError of 0.011.



(c) The elliptical diagram from **eulerr**, with a stress of 2.587×10^{-13} and diagError of 9.944×10^{-8}.

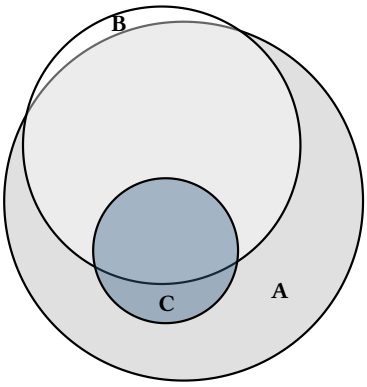
Figure 3.1. A comparison of a Euler diagram generated with **venneuler** with two generated from **eulerr** with circles and ellipses respectively.

Micallef and Rodgers [3] features a diagram from Lenz and Fornoni [26] that they favorably remodelled using **eulerAPE**. We will do the same here, using **eulerr** and compare the results of the two packages. The data from the original diagram was

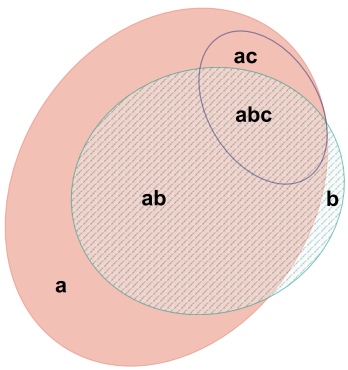
$$\begin{gathered} A = 0.36, \quad B = 0.03, \quad C = 0, \\ A\&B = 0.41, \quad A\&C = 0.04, \quad B\&C = 0, \quad \text{and} \\ A\&B\&C = 0.11. \end{gathered}$$

However, because **eulerAPE** cannot fit set configurations with empty intersections, the authors used 0.00001 in place of 0. Since the diagram was fitted with ellipses, we will do the same with **eulerr** but keep the areas as in the original data.

The fits from both packages are exact (Figure 3.2). Although we instructed **eulerr** to allow ellipses in the fit, the algorithm stuck to circles, which, given that the fit is exact, is the appropriate choice since circles are easier to interpret [7]. The reason **eulerAPE** did not is that it tries to keep the three shapes intersecting (albeit marginally), which cannot be done with circles if the layout is to be exact.



(a) The diagram from **eulerr**.



(b) The diagram from **eulerAPE**.

Figure 3.2. Diagrams from **eulerAPE** and **eulerr** based on data from a diagram in Lenz and Fornoni [26]. The diagram from **eulerAPE** has been modified to enlarge fonts and remove labels for *b* and *bc*, which had been added at seemingly arbitrary locations in the original diagram. Both diagrams are exact.

	Amelie	Pulp Fiction	Miss Congeniality	Armageddon	Rashomon	Coyote Ugly
Amelie	38,753					
Pulp Fiction	15,197	70,153				
Miss Congeniality	1,829	3,854	37,837			
Armageddon	1,218	6,593	10,536	40,345		
Rashomon	2,087	2,799	132	143	6,209	
Coyote Ugly	610	2,206	5,965	5,699	38	15,611

Table 3.1. The data from the Netflix Prize dataset used by Frederickson [27]. Each number indicates the number of users who gave the corresponding movies a five-star rating.

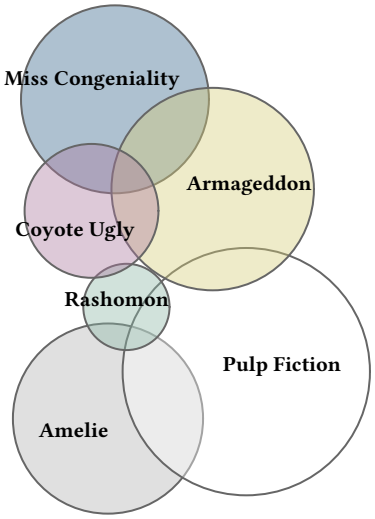
To conclude our case studies, we turn to a diagram that was featured on the website of the author of **venn.js** [27]. The diagram is based on the *Netflix Prize Dataset*[22], from which the author

> [...] picked 6 movies, kind of at random – and then represented them using the set of users that gave the movie a 5 star rating.

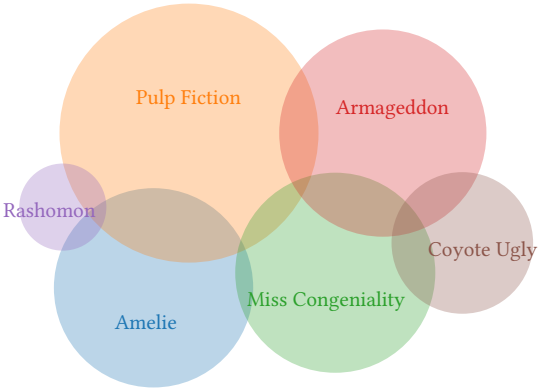
The movies were *Amelie, Pulp Fiction, Miss Congeniality, Armageddon, Rashomon,* and *Coyote Ugly*. The data only includes pairwise relationships (Table 3.1).

The fit from **eulerr** is marginally better than that of **venn.js**. The stress and diagError of the **eulerr** diagram (Figure 3.4a) are 0.003 and 0.014 respectively, whilst the same figures are 0.004 and 0.015 for the **venn.js** diagram (Figure 3.4b).

[22] The dataset is no longer available on this webpage, but has been archived at `https://www.kaggle.com/netflix-inc/netflix-prize-data` for those who are interested.



(a) The fit from **eulerr** with a stress of 0.003 and diagError of 0.014.



(b) The fit from **venn.js** with a stress of 0.004 and diagError of 0.015.

Figure 3.4. The Euler diagrams generated from the Netflix Prize data. The difference in the two layouts primarily concern the placements of *Rashomon* and *Miss Congeniality*.

3.2 Consistency

To compare the consistency among **eulerr**, **venneuler**, **eulerAPE**, **venn.js**, and **Vennerable**, we generate random diagrams of circles and ellipses (separately), compute their areas, and attempt to reproduce the original diagrams using the various software. We restrict ourselves to diagrams consisting of between three and eight shapes.

For the circles, we sample radii (r_i) and coordinates (h_i and k_i) from

$$\begin{aligned} h_i, k_i &\sim \mathcal{U}(0, 1) \\ r_i &\sim \mathcal{U}(0.2, 0.6). \end{aligned} \tag{3.1}$$

For the ellipses, we sample semiaxes (a_i and b_i), coordinates (h_i and k_i), and rotation axes (ϕ_i) from

$$\begin{aligned} h_i, k_i &\sim \mathcal{U}(0, 1) \\ r_i &\sim \mathcal{U}(0.2, 0.6) \\ c_i &\sim \mathcal{U}(0.5, 2) \\ a_i &= c_i r_i \\ b_i &= \frac{r_i}{c_i} \\ \phi_i &\sim \mathcal{U}(0, \pi). \end{aligned} \tag{3.2}$$

Next, we compute the required areas, ω (Definition 2.1) and fit Euler diagrams using the aforementioned packages. Finally, we compute and return diagError (2.7) and score each diagram as a success if its diagError is lower than 0.01, that is, if no portion of the diagram is one *percentage point* off from that of the input; note that this is always achievable since our input comes from sampled diagrams.

For each number of shapes, $N = 3, 4, \ldots, 8$, we run the simulations until we have achieved a 95% confidence interval around $\hat{p}$, the proportion of successful diagrams, no wider than 0, and for a minimum of 1,000 iterations. We use the normal approximation interval,

$$\mathcal{I}(\hat{p})_{0.95} = \hat{p} \pm z_{0.95} \sqrt{\frac{\hat{p}(\hat{p} - 1)}{n}}. \tag{3.3}$$

The procedure is formalized in Algorithm 1. For circular diagrams, **eulerr** outperforms **Vennerable** and **venneuler** in consistency by considerable margin and is on par with **venn.js** and **eulerAPE** (Figure 3.5). **eulerr** and **eulerAPE** are the only packages that successfully fits all of the circular diagrams (although **eulerAPE** only accepts a subset of our three-set diagrams). **venn.js** fails in 4 cases out of 6000.

For ellipses of three shapes, **eulerr** successfully fits all diagrams whilst **eulerAPE** fails in 6 cases out of 1000.

For ellipses of four or more sets—which only **eulerr** accepts—the consistency drops for each additional set, from 78% at four sets to 38% at eight sets. **Vennerable**, which is only able to produce

for $N \leftarrow 3 : 8$ **do**
  i, successes $\leftarrow 0$
  do
    $i \leftarrow i + 1$
    $\alpha_i \leftarrow$ RANDOM DIAGRAM
    $\omega_i \leftarrow$ COMPUTE OVERLAPS(α_i)
    for all $j \leftarrow$ software **do**
      $\hat{p}_j \leftarrow \text{successes}_j / i$
      if $\mathcal{I}(\hat{p})_{0.95}$ is wider than 0.02 **or** $i \leq 1000$ **then**
        $A_{i_j} \leftarrow$ FIT DIAGRAM(ω_i)
        if DIAGERROR$(A_{i_j}, \omega_i) < 0.01$ **then**
          $\text{successes}_j \leftarrow \text{successes}_j + 1$
  while all $\mathcal{I}(\hat{p})_{0.95}$ are wider than 0.02 **or** $i \leq 1000$

Algorithm 1. The algorithm used to simulate diagrams of circles or ellipses, reverse-engineer set relationships, and fit Euler diagrams to these relationships using the different software packages. $\mathcal{I}(\hat{p})_{0.95}$ is the binomial proportion confidence interval (3.3).

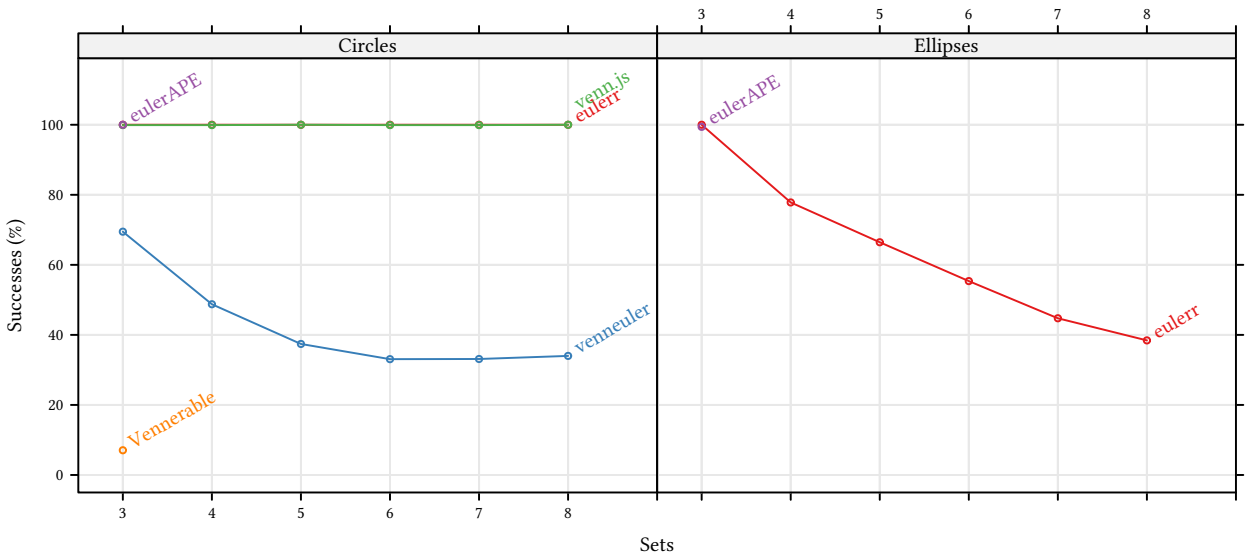


Figure 3.5. Proportions of successfully reproduced Euler diagrams from set configurations based on sampled circular and elliptical diagrams generated from the distributions in (3.1) and (3.2). The results are based on at least 1000 iterations for each software package and number of sets and have at most a 0.02-wide symmetric 0.95% confidence interval around the displayed point estimate. A success is defined as a diagram with a diagError (2.7) below 0.01.

three-set diagrams, only produces accurate diagrams for 7% of the random layouts and moreover fails with an error in 6 cases.

3.3 Accuracy

In Section 3.2, we assessed the efficacy in reproducing diagrams with exact, albeit unknown, solutions. Reality, however, often presents us with relationships that lack such exact solutions. Yet, when there is no exact solution, one might still exist that is *good enough*, which we naturally want our model to find.

To assess the accuracy in producing diagrams for which there might not be an exact fit, we generate random set relationships, fiting Euler diagrams to each using the software under study. For each $N = 3, 4, \ldots, 8$ sets, we initialize N-set combinations with all items set to 0. For every set relationship, we pick N elements from the relationship at random—making sure that each set is selected at least once—and assign to each a number generated from $\mathcal{U}(\epsilon, 1)$, where ϵ is defined as the square root of the current machine's smallest representable difference between one and the smallest value greater than one[23]. Next, we draw a sample of a random subset of the remaining $2^N - 1 - N$ set intersections and assign to them a number from $\mathcal{U}(\epsilon, 1)$ as before.

[23] The R-code used to generate this ϵ is `sqrt(.Machine$double.eps)`

We run our simulations for a minimum of 1,000 iterations and until we achieve a 95% confidence interval around the mean diagError no wider than 0.02. The confidence level used is based on the t-distribution,

$$\mathcal{I}(\bar{x})_{0.95} = \bar{x} \pm t_{0.95} \frac{s}{\sqrt{n}}, \tag{3.4}$$

where

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^{n} (x - \bar{x})^2}.$$

The procedure is formalized in Algorithm 2.

for $N \leftarrow 3 : 8$ **do**
  $i \leftarrow 0$
  do
    $i \leftarrow i + 1$
    for $k \leftarrow 1 : N$ **do**
      random element in $\{\omega_i : \omega_i \cap F_k \neq \emptyset\} \leftarrow \mathcal{U}(\epsilon, 1)$
    for all $j \leftarrow$ software packages **do**
      $A_{j_i} \leftarrow$ FIT DIAGRAM(ω_i)
      $x_{j_i} \leftarrow$ DIAGERROR(A_{j_i}, ω_i)
  while all $\mathcal{I}(\bar{x})_{0.95}$ are wider than 0.02 **or** $i \leq 1000$

Algorithm 2. The algorithm used to simulate and fit random set relationships to assess the accuracy of the various software we are studying. $\mathcal{I}(\bar{x})_{0.95}$ is the 95% confidence interval around the mean diagError (3.4), F denotes a set, and ϵ is the square root of the difference between 1 and the least value greater than 1 on our machine.

However, given that neither **Vennerable** nor **eulerAPE** are capable of fitting Euler diagrams that feature subset or disjoint relationships, and because fully intersecting diagrams are more difficult

to fit—at least with circles—we run a separate treatment in which we modify Algorithm 2 so that every intersection is initialized to a value in $\mathcal{U}(\epsilon, 1)$.

From the results generated via Algorithm 2, we can surmise that **eulerr**'s elliptical diagrams achieve the lowest median stress and diagError (Figure 3.6) for all set sizes, although the difference is relatively more pronounced for set sizes three and four.

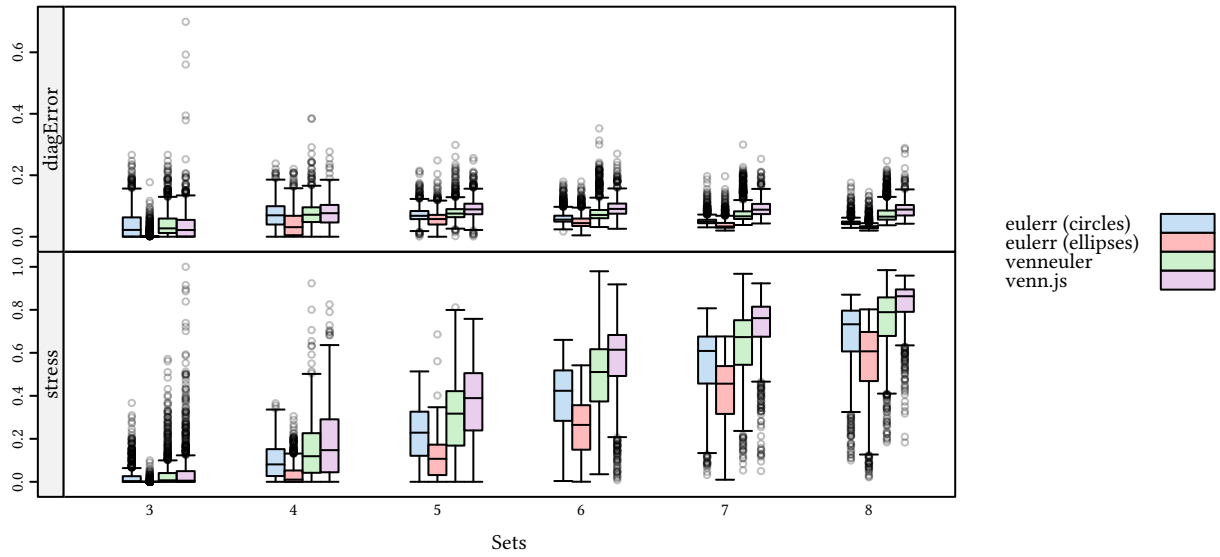


Figure 3.6. Tukey box plots of Euler diagrams based on set relationships that may or may not have perfect solutions, generated from (3.2).

We also find that among the algorithms that fit circular Euler diagrams, **eulerr** offers the lowest median stress across all the sizes of set relationships. For diagError, however, **venn.js** produces diagrams with the least loss for three sets, whilst **eulerr** scores better for all the remaining set sizes, for which **venn.js** is moreover outdone by **venneuler**.

Looking at the results from the simulation of three-set relationships with all intersections present (Figure 3.7), **eulerr** still produces the most accurate diagrams provided that we use ellipses—both in terms of stress and diagError. The difference next to **eulerAPE** is neglible, however, with differences below 10^{-7}.

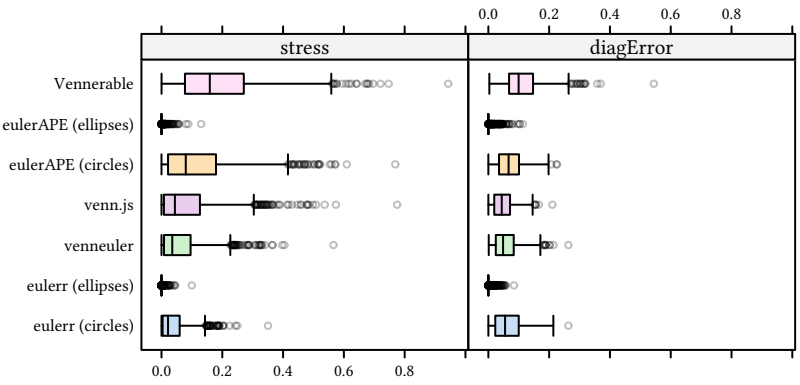


Figure 3.7. Tukey box plots of diagError and stress for Euler diagrams based on set relationships of three sets with every intersection present.

For circular diagrams, **venn.js** achieves the lowest median diagError at 0.044, followed by **venneuler** at 0.048, **eulerr** at 0.055, **eulerAPE** at 0.067, and **Vennerable** at 0.1. As for stress, however, the order is partially reversed with respective stress values at 0.022, 0.035, 0.044, 0.08, 0.159 for **eulerr**, **venneuler**, **venn.js**, **eulerAPE**, and **Vennerable** respectively.

3.4 Performance

Using the same procedure as in Algorithm 2, we generate random set relationships and measure the time it takes for each software package to form a diagram from the input. We use **microbenchmark** [25] for the benchmarks, which randomizes the order in which the packages are called between trials. Because neither the current version of **venn.js**, nor any version of **eulerAPE**, have been implemented in R, we omit these packages from the performance benchmarks.

eulerr is faster for circular diagrams up to and including seven sets (Figure 3.8), after which **venneuler** catches up and subsequently outperforms **eulerr** for eight-set relationships. For a three-set diagram, for instance, **eulerr** takes a median of 0.007 seconds to find a fit, whilst **venneuler** takes 0.414 seconds, and **Vennerable** 0.056 seconds. For eight sets, meanwhile, **eulerr** takes 2.68 seconds and **venneuler** 2.059.

The computation time for the elliptical diagrams from **eulerr** generally vary more but are still, on average, faster than **venneuler** for up to five sets—albeit with the exception of diagrams of three sets, where the resulting bimodal distribution is a consequence of the time-consuming last-ditch optimizer (Section 2.3.4) that is activated by default if the fit is not error-free after the final optimizer.

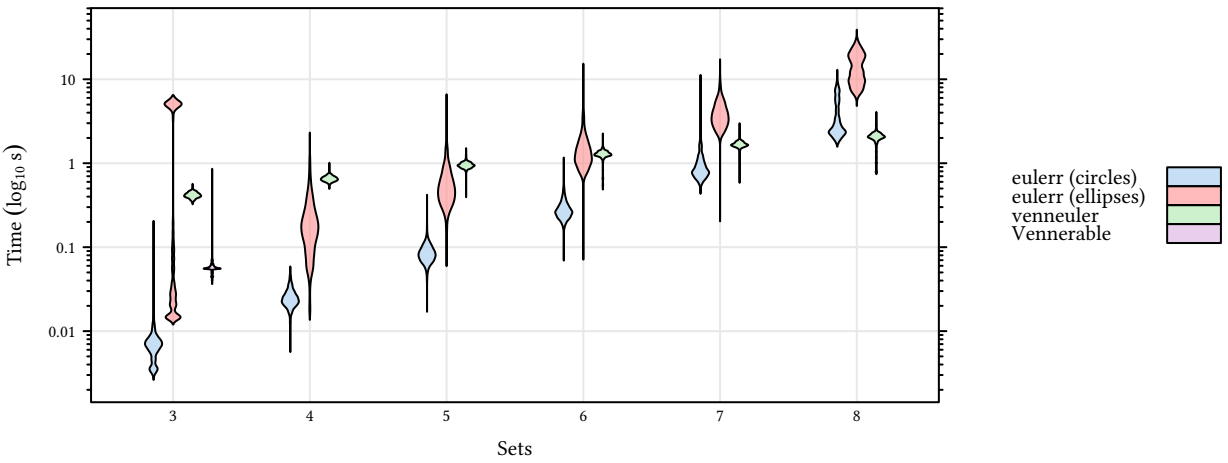


Figure 3.8. Violin plot of the performance of **eulerr**, **venneuler**, and **Vennerable** on random set relationships of three to eight sets. The density smoother is that of Sheather and Jones [28].

4 Discussion

In this thesis, we have presented a novel method for generating elliptical Euler diagrams for any number of sets. We have shown that the package is both more consistent and accurate than the other packages analyzed as well as faster for up to seven sets.

eulerr reproduced all the circular diagrams generated in Section 3.2 within our stipulated margin of error. No other package managed this task, although the failing diagrams of both **eulerAPE** and **venn.js** numbered in the single digits. **venneuler**, in contrast, was not able to adequately reproduce 57% of these diagrams, which, however, was still better than the results on a previous test [29].

Some of the examined set relationships feature disjoint or subset relationships, which **venneuler** have problems fitting on account of its initial optimizer. As we discussed in Section 1.1, **eulerr**, **venn.js**, and **venneuler** all use multi-dimensional scaling for their initial diagram. **venneuler**, however, unnecessarily restricts the locations of disjoint and subset circles. Given, for instance, two disjoint sets, the algorithm will attempt to place them tangent to one another; otherwise, the optimizer will report loss. Likewise, for a subset relationship, **venneuler** tries to place the smaller circle at the exact midpoint of the enclosing shape.

For the fit to be accurate, however, those restrictions are pointless as long as the sets remain disjoint or subset. For **venneuler**, this becomes problematic because the required positions might interfere with space that could be used by other sets to improve the fit. The constrained multi-dimensional scaling method in **venn.js** and **eulerr** circumvents this by assigning a loss and gradient of zero when the pairwise set intersection *and* the candidate circles are disjoint or subset. This makes it easier for the starting layout to find a good initial layout.

Another reason for the mediocre results of **venneuler** could be both that the optimizer terminates prematurely in case the relative reduction in stress is considered negligible between iterations *or* that the number of iterations is too low.

eulerr managed to also reproduce all of the elliptical three-set diagrams perfectly but failed for an increasing number of cases as we added sets to our input. For three-set diagrams, **eulerr** employs a rigorous last-ditch optimizer in case the fit it not adequate after the first step of the final optimization procedure. This step is time-consuming (as we saw in Section 3.4), but if activated would yield better results also for sets of more than three shapes.

The elliptical diagrams from **eulerr** are more accuracte for set relationships that might lack exact solutions compared with the other packages in this thesis, although the difference next to **eulerAPE** is slight. Elliptical Euler diagrams are more successful since they allow two additional degrees of freedom for each set, that is, rotation and stretching. The marginal gains are greatest for three sets and diminishes as we add more sets. This is what we expect: complicated inputs require complicated models and there are many set configurations that even elliptical Euler diagrams cannot support—consider, for instance, the complicated geometry of the Venn diagram in Figure 4.1 that is required to represent all the intersections between six sets.

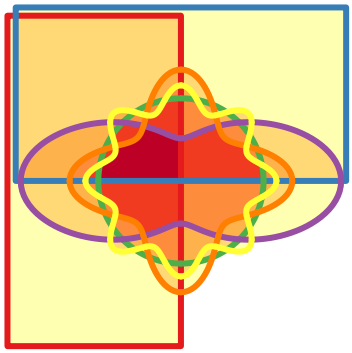


Figure 4.1. A six-set Venn diagram using Edward's method [30]; this diagram was generated with **Vennerable**.

Considering only circular Euler diagrams, **eulerr** remains the best choice for both diagError and stress in all cases except for three-set diagrams, where it ranks best in terms of stress but not diagError, which **venn.js** scores best on. It might be appropriate to note, however, that **eulerr**, **venneuler**, and **venn.js** all try to minimize the residual sums of squares in one way or another—*not* diagError. In essence, this means that the lower diagError from **venn.js**, regardless of whether it is desired, is not a sign that the algorithm performed as intended, particularly not when stress is considered as well[24].

[24] Stress, on the other hand, *is* directly related to the residual sums of squares.

Surprisingly, given the results in Section 3.2, **venn.js** performs worse than **venneuler** in all of the remaining cases. We can only speculate as to reasons for this, but it is possible that the Nelder–Mead variant used in **venn.js** chokes on the more complicated layouts; indeed, this was our experience when designing **eulerr**, which at one point featured a version of the same optimizer. In our case, `nlm()` from the **stats** package was found to be superior.

In contrast to **eulerAPE**'s elliptical diagrams, its circular ones performed worse than all other packages save for **Vennerable**. This result likely stems from the particular loss function used in **eulerAPE**, which prohibits empty overlaps in the diagram—often at the expense of the overall fit. The author's argue that this function drives the optimizer away from local minima [3], but we could not find any such issues with **eulerr**. On the other hand, this feature of **eulerAPE** might be desired by those who prioritize the nominal[25], rather than proportional, representativeness of the diagram.

[25] Nominal in the sense that each relationship in the input is also represented in the diagram.

Performance-wise, **eulerr** is faster than both **Vennerable** and **venneuler** for up to seven sets. One reason for this is likely the implementation in C++ and use of the Armadillo library [31] provided by the interfaces **Rcpp** [32] and **RcppArmadillo** [33]. Another reason—possibly the foremost—comes from the exact-area algorithm that involves less work than the quadtrees of **venneuler** with few sets.

Paradoxically, this is also why the performance of **eulerr** suffers as the number of sets increase. In essence, we must examine every possible intersection when computing the areas. For eight sets, for

instance, this means investigating $2^8 - 1 = 255$ intersections. Asymptotically, the algorithm thus converges in $O(2^n)$ time. Wilkinson [5], meanwhile, report convergence for **venneuler** in $O(n)$ time, which there indeed is evidence of in Figure 3.8. As a result, the computational demand of complicated set configurations make fitting difficult diagrams with **eulerr** prohibitive where speed is a concern. On the other hand, Euler diagrams seldom make sense for such complicated set relationships—only rarely will they conform to an adequate fit. Nevertheless, future versions of this algorithm should consider implementing approximate area-calculations when the number of sets is large to cater to the, albeit few, instances where a Euler diagram is appropriate.

Fitting diagrams is substantially harder with ellipses than with circles. There are many more local minima that might hobble our optimizer and since our initial configuration only considers circles, such minima cannot reliably be avoided before the final optimization step. And because we default to a local optimizer in this final step, it is not uncommon that we terminate before finding the global minimum. The last-ditch optimizer battles such minima with brute force: it tries a vast number of permutations and uses the previous fits (initial and semi-final) only to set up constraints for the algorithm. The downside to using this algorithm is that it, in contrast to human beings, does not favor circles over ellipses, which means that we might get elliptical diagrams when circular ones would do. Future efforts in this field should consider penalization, or similar techniques, in order to promote user-friendly diagrams.

4.1 Conclusion

In all of the scenarios examined in this thesis, **eulerr**'s elliptical Euler diagrams offer solutions with the least error among all of the packages tested. For three sets, **eulerr**'s accuracy is equalled only by the elliptical diagrams from **eulerAPE**, which, however, impose restrictions that **eulerr** do not, namely, that there be no disjoint or subset relationships.

eulerAPE's restriction to three sets is discussed by the authors of the package, who motivate this limitation with the propensity of Euler diagrams with more sets to lack adequate solutions and that their complexity make implementations difficult [9]. Whilst it is true that inputs with more than three sets do not always reduce to adequate Euler diagrams, it is our stance that those that *do*, warrant a software implementation that enables users to find them, given that Euler diagrams are intuitive visualizations that are easily grasped by most viewers.

The foremost shortcoming of **eulerr** is its failure to consistently find optimal elliptical diagrams, which is evident in Section 3.2, wherein a portion of the sampled diagrams are not refit adequately, implying that the accuracy (see Section 3.3) must have potential to

improve. This problem is not intractable and we believe it could be overcome either by

- relying on brute-force global optimizers that thoroughly examine the search space and attempt to optimize—possibly parallelize—these routines to make complex diagrams fit with reasonable speed, or by
- designing an algorithm for the initial configuration that works specifically for ellipses and avoids local minima ahead of final optimization.

Whichever direction future research takes, we believe that the advances presented in this thesis serves as another step in the direction towards more accurate Euler diagrams.

Appendices

A Visualization

Once we have ascertained that our Euler diagram fits well, we can turn to visualizing the solution. For this purpose, **eulerr** leverages the **Lattice** graphics system [34] for R to offer intuitive and granular control over the output.

Plotting the ellipses is straightforward using the parametrization of a rotated ellipse,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} h + a\cos\theta \\ k + b\sin\theta \end{bmatrix}, \quad \text{where } \theta \in [0, 2\pi],\ a, b > 0.$$

Most users will also prefer to label the ellipses and their intersections with text and this, however, is considerably more involved.

A.1 Labeling

Labeling the ellipses is complicated since the shapes of the intersections often are irregular, lacking well-defined centers; we know of no analytical solution to this problem. As usual, however, the next-best option turns out to be a numerical one. First, we locate a point that is inside the required region by spreading points across the discs involved in the set intersection. To distribute the points, we use a modification of *Vogel's method* [35, 36] adapted to ellipses. Vogel's method spreads points across a disc using

$$p_k = \begin{bmatrix} \rho_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} r\sqrt{\frac{k}{n}} \\ \pi(3 - \sqrt{5})(k - 1) \end{bmatrix} \quad \text{for } k = 1, 2, \ldots, n. \tag{A.1}$$

In our modification, we scale, rotate, and translate the points formed in (A.1) to match the candidate ellipse. We rely, as before, on projective geometry to carry out the transformations in one go:

$$p' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix},$$

where h, k translates, ϕ rotates, and a, b stretches the ellipse.

After we spread our points throughout the ellipse and find a point, p_i', that is contained in our desired intersection, we proceed to optimize its position numerically. The position we are looking for is

that which maximizes the distance to the closest ellipse in our diagram to provide as much margin as possible for the label. This is a maximization problem with a loss function equal to

$$\mathcal{L}(x, y) = \min_{i=1,2,\ldots,N} f(x, y, h_i, k_i, a_i, b_i, \phi_i) \tag{A.2}$$

where f is the function that determines the distance from a point (x, y) to the ellipse defined by h, k, a, b and ϕ.

Similarly to fitting Euler diagrams in the general case, there appears to be no analytical solution to computing the distance from a point to an ellipse. The numerical solution we use has been described by Eberly [37] and involves solving the roots to a quartic polynomial via a robust bisection optimizer.

To optimize the location of the label, we employ a version of the *Nelder–Mead method* [38], which has been translated from a Matlab code by Kelley [39] and adapted for **eulerr** to ensure that it converges quickly and that the simplex remains within the intersection boundaries (since we want the local maximum). The method is visualized in Figure A.1.

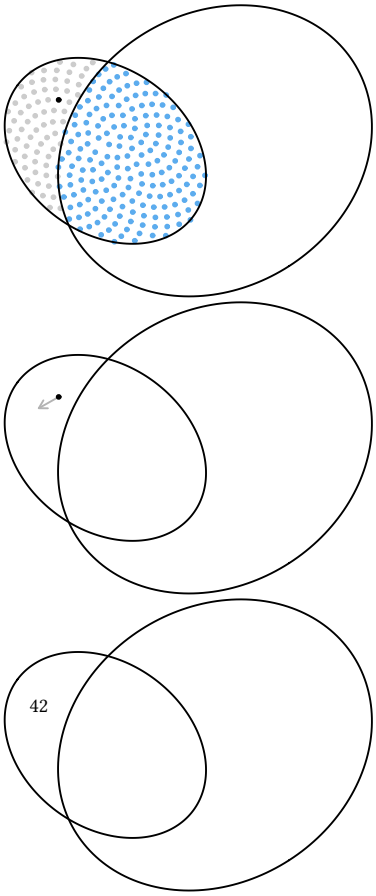


Figure A.1. The method eulerr uses to locate an optimal position for a label in three steps from top to bottom: first, we spread sample points on one of the ellipses and pick one inside the intersection of interest, then we begin moving it numerically, and finally place our label.

A.2 Aesthetics

Euler diagrams display both quantitative and qualitative data. The quantitative aspect is the quantities or sizes of the sets depicted in the diagram and is visualized by the relative sizes, and possibly the labels, of the areas of the shapes—this is the main focus of this paper. The qualitative aspects, meanwhile, consist of the mapping of each set to some quality or category, such as having a certain gene or not. In the diagram, these qualities can be separated through any of the following aesthetics:

- color,
- border type,
- text labelling,
- transperancy,
- patterns,

or a combination of these. The main purpose of these aethetics is to separate out the different ellipses so that the audience may interpret the diagram with ease and clarity.

Among these aesthetics, the best choice (from a viewer perspective) appears to be color [7], which provides useful information without extraneous chart junk [40]. The issue with color, however, is that it cannot be perceived perfectly by all—8% of men and 0.4% of women in European Caucasian countries, for instance, suffer the most common form, red–green color deficiency [41]. Moreover, color is

often printed at a premium in scientific publications and adds no information to a diagram of two shapes.

For these reasons, **eulerr** defaults to distinguishing ellipses with color using a color palette generated via the R package **qualpalr** [42], which automatically generates qualitative color palettes based on a perceptual model of color vision that optionally caters to color vision deficiency. This palette has been manually modified to fullfil our other objectives of avoiding using colors for two sets. The first eight colors of the pallete are visualized in Figure A.2.

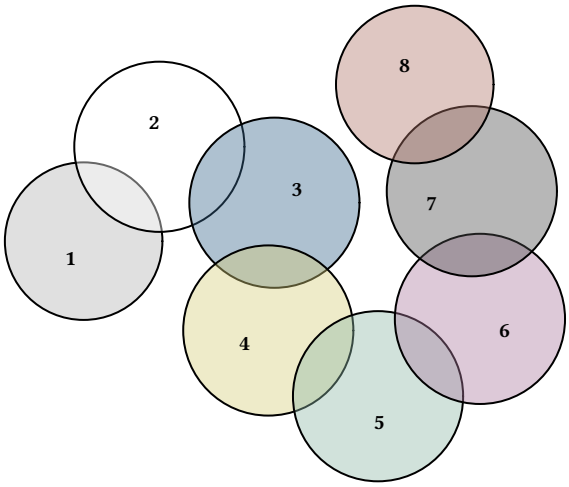


Figure A.2. The eight first colors of the default color palette.

A.3 Normalizing dispered layouts

If there are disjoint clusters of ellipses, the optimizer will often spread these out more than is necessary, wasting space in our diagram. To tackle this, we use a SKYLINE-BL rectangle packing algorithm [43] designed specifically for **eulerr**. In it, we surround each ellipse cluster with a bounding box, pack these boxes into a bin of appropriate size and aspect ratio, and adjust the coordinates of the ellipses in the clusters to compact our diagram. As a bonus, this increases the chance of having similar layouts for different function calls.

B Usage

`euler()` and `plot()` are the only functions that users of **eulerr** need concern themselves with. In Section 2.1, we described the various forms of input that the function can be supplied with. Using the first form, we will showcase how a Euler diagram is fit. For this example, we use a diagram from a publication by Junta et al. [44] that was also tackled by Wilkinson [5]. We load the package, specify our diagram, and fit it using `euler()` as follows.

```
library(eulerr)
junta_2009 <- c("SE" = 13, "Treat" = 28, "Anti-CCP" = 101,
                "DAS28" = 91, "SE&Treat" = 1, "SE&DAS28" = 14,
                "Treat&Anti-CCP" = 6, "SE&Anti-CCP&DAS28" = 1)
fit1 <- euler(junta_2009)
```

Printing the results provides a summary of the fit, including the stress and diagError metrics that were introduced in Section 2.4.

```
fit1 # or equivalently print(fit1)


##                          original fitted residuals regionError
## SE                              13 12.780     0.220       0.000
## Treat                           28 27.525     0.475       0.000
## Anti-CCP                       101 99.287     1.713       0.002
## DAS28                           91 89.457     1.543       0.001
## SE&Treat                         1  0.983     0.017       0.000
## SE&Anti-CCP                      0  0.000     0.000       0.000
## SE&DAS28                        14 13.763     0.237       0.000
## Treat&Anti-CCP                   6  5.898     0.102       0.000
## Treat&DAS28                      0  0.000     0.000       0.000
## Anti-CCP&DAS28                   0  0.000     0.000       0.000
## SE&Treat&Anti-CCP                0  0.000     0.000       0.000
## SE&Treat&DAS28                   0  0.000     0.000       0.000
## SE&Anti-CCP&DAS28                1  0.000     1.000       0.004
## Treat&Anti-CCP&DAS28             0  0.000     0.000       0.000
## SE&Treat&Anti-CCP&DAS28          0  0.000     0.000       0.000
##
## diagError: 0.004
## stress:    0
```

The fit is more or less equivalent to that of **venneuler** [5]. There *is* an error but it is small at a diagError of 0.004. We could, however, try to improve the fit using ellipses:

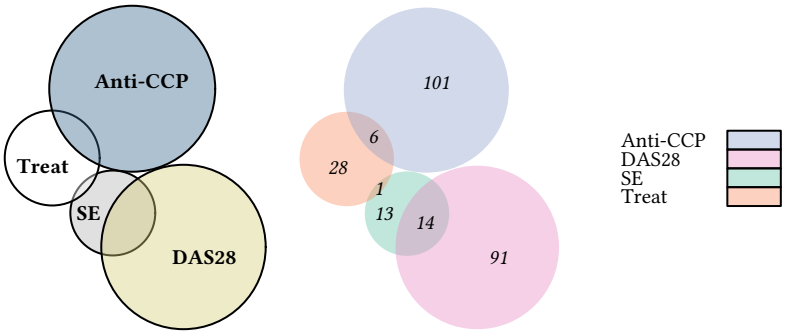
```
# Fit the data using ellipses instead
fit2 <- euler(junta_2009, shape = "ellipse")

# Compare the fits on diagerror
fit1$diagError - fit2$diagError


## [1] 0
```

Comparing the two fits in diagError, however, shows that we have not bettered the fit in any meaningful way. Our next goal is to visualize the layout, which we do both using the default options and by customizing the fit, adding quantities, replacing the sets' labels with a key, removing lines, and changing the fills (Figure B.1) using the **RColorBrewer** package [45].

```
p1 <- plot(fit1)
p2 <- plot(fit1,
           quantities = list(fontface = 3),
           fill = RColorBrewer::brewer.pal(4, "Set2"),
           border = "transparent",
           auto.key = list(space = "right")) # key on the right
```



(a) The default settings.

(b) Custom plot settings.

Figure B.1. The same fit visualized in two distinct ways.

Bibliography

[1] Leonhard Euler. *Letters of Euler to a German princess, on different subjects in physics and philosophy*. Murray and Highley, 1802. URL http://archive.org/details/letterseulertoa00eulegoog.

[2] Jonathan Swinton. *Vennerable: Venn and Euler area-proportional diagrams*, 2011. URL https://github.com/js229/Vennerable. R package version 3.1.0.9000.

[3] Luana Micallef and Peter Rodgers. eulerAPE: drawing area-proportional 3-Venn diagrams using ellipses. *PLOS ONE*, 9(7):e101717, July 2014. ISSN 1932-6203. doi: 10.1371/journal.pone.0101717.

[4] Luana Micallef and Peter Rodgers. eulerForce: force-directed layout for Euler diagrams. *Journal of Visual Languages and Computing*, 25(6): 924–934, December 2014. ISSN 1045-926X. doi: 10.1016/j.jvlc.2014.09.002.

[5] L. Wilkinson. Exact and approximate area-proportional circular Venn and Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):321–331, February 2012. ISSN 1077-2626. doi: 10.1109/TVCG.2011.56.

[6] Hans A. Kestler, André Müller, Johann M. Kraus, Malte Buchholz, Thomas M. Gress, Hongfang Liu, David W. Kane, Barry R. Zeeberg, and John N. Weinstein. VennMaster: area-proportional Euler diagrams for functional GO analysis of microarrays. *BMC Bioinformatics*, 9:67, January 2008. ISSN 1471-2105. doi: 10.1186/1471-2105-9-67.

[7] Andrew Blake. *The impact of graphical choices on the perception of Euler diagrams*. Ph.D. dissertation, Brighton University, Brighton, UK, February 2016. URL http://eprints.brighton.ac.uk/15754/1/main.pdf.

[8] Stirling Christopher Chow. *Generating and drawing area-proportional Euler and Venn diagrams*. Ph.D. dissertation, University of Victoria, Victoria, BC, Canada, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.2678&rep=rep1&type=pdf.

[9] Luana Micallef. *Visualizing set relations and cardinalities using Venn and Euler diagrams*. Ph.D. dissertation, University of Kent, September 2013. URL https://kar.kent.ac.uk/47958/.

[10] Ben Frederickson. venn.js: area proportional Venn and Euler diagrams in JavaScript, November 2016. URL https://github.com/benfred/venn.js. original-date: 2013-05-09T17:13:20Z.

[11] David Eberly. The area of intersecting ellipses, November 2016. URL https://www.geometrictools.com/Documentation/AreaIntersectingEllipses.pdf.

[12] Jürgen Richter-Gebert. *Perspectives on Projective Geometry: A Guided Tour Through Real and Complex Geometry*. Springer, Berlin, Germany, 1 edition, February 2011. ISBN 978-3-642-17286-1.

[13] Stirling Chow and Peter Rodgers. Constructing area-proportional Venn and Euler diagrams with three circles. In *Euler Diagrams Workshop 2005*, pages 182–196, August 2005. URL http://www.cs.kent.ac.uk/pubs/2005/2354.

[14] Ben Frederickson. A better algorithm for area proportional venn and euler diagrams, June 2015. URL http://www.benfrederickson.com/better-venn-diagrams/.

[15] Robert B. Schnabel, John E. Koonatz, and Barry E. Weiss. A modular system of algorithms for unconstrained minimization. *ACM Trans Math Softw*, 11 (4):419–440, December 1985. ISSN 0098-3500. doi: 10.1145/6187.6192.

[16] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL https://www.R-project.org/.

[17] John C. Nash. *Nonlinear parameter optimization using R tools*. Wiley, Chichester, West Sussex, 1 edition, May 2014. ISBN 978-1-118-56928-3.

[18] Darel R. Finley. Ultra-easy algorithm with C code sample. http://alienryderflex.com/polygon_area/, December 2006. URL http://alienryderflex.com/polygon_area/.

[19] Dirk Eddelbuettel. *RcppDE: Global Optimization by Differential Evolution in C++*, 2016. URL https://CRAN.R-project.org/package=RcppDE. R package version 0.1.5.

[20] Katherine M. Mullen, David Ardia, David L. Gil, Donald Windover, and James Cline. DEoptim: An R package for global optimization by differential Evolution. *Journal of Statistical Software*, 40(6), April 2011. doi: 10.18637/jss.v040.i06. URL https://www.jstatsoft.org/article/view/v040i06.

[21] Yang Xiang, Sylvain Gubian, Brian Suomela, and Julia Hoeng. Generalized simulated annealing for global optimization: the GenSA package. *The R Journal*, 5(1):13–28, June 2013. URL https://journal.r-project.org/archive/2013/RJ-2013-002/index.html.

[22] R Core Team. The Comprehensive R Archive Network, November 2017. URL https://CRAN.R-project.org/.

[23] Hadley Wickham and Winston Chang. *devtools: Tools to Make Developing R Packages Easier*, 2017. URL https://CRAN.R-project.org/package=devtools. R package version 1.13.4.

[24] Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. *shiny: Web Application Framework for R*, 2017. URL https://CRAN.R-project.org/package=shiny. R package version 1.0.5.

[25] Jeroen Ooms. *V8: Embedded JavaScript Engine for R*, 2017. URL https://CRAN.R-project.org/package=V8. R package version 1.5.

[26] Oliver Lenz and Alessia Fornoni. Chronic kidney disease care delivered by US family medicine and internal medicine trainees: results from an online survey. *BMC medicine*, 4:30, December 2006. ISSN 1741-7015. doi: 10.1186/1741-7015-4-30.

[27] Ben Frederickson. Venn diagrams with D3.js, June 2015. URL http://www.benfrederickson.com/venn-diagrams-with-d3.js/.

[28] Simon Sheather and M. C. Jones. A Reliable Data-Based Bandwidth Selection Method for Kernel Density Estimation. *Journal of the Royal Statistical Society*, 53(3):683–690, January 1991. doi: 10.2307/2345597. DOI: 10.2307/2345597.

[29] Ben Frederickson. venn.js compared to venneuler, June 2015. URL http://benfred.github.io/venn.js/tests/venneuler_comparison/.

[30] A. W. F. Edwards. *Cogwheels of the Mind: The Story of Venn Diagrams*. Johns Hopkins University Press, Baltimore, USA, 1 edition, April 2014. ISBN 978-0-8018-7434-5. URL https://jhupbooks.press.jhu.edu/content/cogwheels-mind.

[31] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *The Journal of Open Source Software*, 1(2): 26, 2016. doi: 10.21105/joss.00026. URL http://joss.theoj.org/papers/10.21105/joss.00026.

[32] Dirk Eddelbuettel and Romain François. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8):1–18, 2011. URL http://www.jstatsoft.org/v40/i08/.

[33] Dirk Eddelbuettel and Conrad Sanderson. RcppArmadillo: accelerating R with high-performance C++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. doi: 10.1016/j.csda.2013.02.005.

[34] Deepayan Sarkar. *Lattice: multivariate data visualization with R*. Use R! Springer, New York, USA, 1 edition, 2008. ISBN 978-0-387-75968-5. URL http://www.springer.com/us/book/9780387759685.

[35] Mary K. Arthur. Point picking and distributing on the disc the sphere. Final ARL-TR-7333, US Army Research Laboratory, Weapons and Materials Research Directorate, Abedeen, USA, July 2015. URL www.dtic.mil/get-tr-doc/pdf?AD=ADA626479.

[36] H. Vogel. A better way to construct the sunflower head. *Mathematical Biosciences*, 44(3-4):179–189, 1979. doi: 10.1016/0025-5564(79)90080-4.

[37] Eberly. Distance from a point to an ellipse, an ellipsoid, or a hyperellipsoid, November 2016. URL https://www.geometrictools.com/Documentation/DistancePointEllipseEllipsoid.pdf.

[38] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7 (4):308–313, January 1965. ISSN 0010-4620. doi: 10.1093/comjnl/7.4.308. URL https://academic.oup.com/comjnl/article/7/4/308/354237/A-Simplex-Method-for-Function-Minimization.

[39] C. T. Kelley. *Iterative methods for optimization*. Number 18 in Frontiers in applied mathematics. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1 edition, 1999. ISBN 0-89871-433-8.

[40] Edward R. Tufte. *The visual display of quantitative information*. Graphics Press, Cheshire, CT, USA, 2 edition, May 2001. ISBN 978-1-930824-13-3.

[41] Jennifer Birch. Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America. A, Optics, Image Science, and Vision*, 29(3): 313–320, March 2012. ISSN 1520-8532.

[42] Johan Larsson. *qualpalr: Automatic Generation of Qualitative Color Palettes*, 2016. URL https://cran.r-project.org/package=qualpalr. R package version 0.3.1.

[43] Jukka Jylänki. A thousand ways to pack the bin – a practical approach to two-dimensional rectangle bin packing, February 2010. URL http://clb.demon.fi/files/RectangleBinPack.pdf.

[44] Cristina Moraes Junta, Paula Sandrin-Garcia, Ana Lúcia Fachin-Saltoratto, Stephano Spanó Mello, Renê D. R. Oliveira, Diane Meyre Rassi, Silvana Giuliatti, Elza Tiemi Sakamoto-Hojo, Paulo Louzada-Junior, Eduardo Antonio Donadi, and Geraldo A. S. Passos. Differential gene expression of peripheral blood mononuclear cells from rheumatoid arthritis patients may discriminate immunogenetic, pathogenic and treatment features. *Immunology*, 127(3):365–372, July 2009. ISSN 1365-2567. doi: 10.1111/j.1365-2567.2008.03005.x.

[45] Erich Neuwirth. *RColorBrewer: ColorBrewer Palettes*, 2014. URL https://CRAN.R-project.org/package=RColorBrewer. R package version 1.1-2.