A Simple Approach for Boundary Improvement of Euler Diagrams

Paolo Simonetto, Daniel Archambault *Member, IEEE*, and Carlos Scheidegger

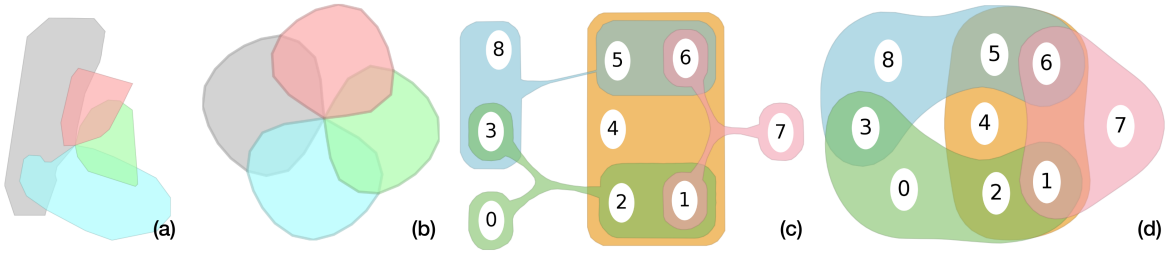

Fig. 1: Boundary smoothing for existing Euler diagrams. (a) General Euler diagram [21, Figure 6b] reproduced. (b) The result of the improvement (c) An Untangled diagram [20, Fig. 5b] reproduced. (d) The result of the improvement.

Abstract— General methods for drawing Euler diagrams tend to generate irregular polygons. Yet, empirical evidence indicates that smoother contours make these diagrams easier to read. In this paper, we present a simple method to smooth the boundaries of any Euler diagram drawing. When refining the diagram, the method must ensure that set elements remain inside their appropriate boundaries and that no region is removed or created in the diagram. Our approach uses a force system that improves the diagram while at the same time ensuring its topological structure does not change. We demonstrate the effectiveness of the approach through case studies and quantitative evaluations.

Index Terms—Euler diagrams, Boundary Improvement, Force-Directed Approaches

1 Introduction

Euler diagrams [12] are an intuitive way to visualize sets and their contents, because they visualize this information in the most straightforward way possible: sets in these diagrams are regions; if the regions overlap, the sets intersect. Recent techniques aim at automating the process of drawing Euler diagrams with closed curves to directly visualize the intersections between sets. Given a list of sets and their intersections, these general techniques produce a drawing for any input instance [3, 9, 20, 21, 24, 25]. Euler diagrams have been applied in various fields, such as bioinformatics [4, 13], digital humanities [32], social networks [18], multimedia database queries [31], and others.

These general techniques are powerful, but the shape of the regions that they generate can often be jagged and elongated due to the large number of constraints that the drawing must satisfy to be valid. At the same time, smoother regions are more aesthetically pleasing, and there is empirical evidence that smooth closed curves pop out visually [5, 14, 19, 30]. Also, early empirical evidence indicates that Euler diagrams are more readable when they have smooth set contours [7]. Thus, an approach able to improve diagrams in general can make these techniques more applicable and practical.

In this paper, we introduce a simple method that can improve the boundaries of any Euler diagram drawing approach (or any other method using polygons and their overlap to indicate the relationship between sets). The approach proposed is a form of *curve shortening flow* for shape improvement [8, 10, 28, 29, 35] applied to the problem of Euler diagram drawing. These techniques are commonplace in geometry processing, graphics, and scientific visualization. However, to the best of our knowledge, curve shortening flow has not been applied to the problem of Euler diagram drawing and is not as well known in information visualization. The essence of our approach is a novel force system based on boundary curvature. This force system can be applied in conjunction with diagram constraints to greatly improve the appearance of the Euler diagrams while ensuring a correct drawing.

The primary contribution of this paper is an algorithm that combines a form of curve shortening flow with Euler diagram drawing methods to improve the smoothness of any Euler diagram drawn with curves. The approach we propose is general and can be applied to the output of all Euler diagram drawing methods (as well as related methods). Our approach is more scalable than existing techniques that improve the smoothness of Euler diagrams. More importantly, the refined drawing has the same topology as the Euler diagram provided as input, and at no point are intersections between the polygons created or destroyed during refinement. Finally, the approach is easy to implement, consisting of only four forces, greatly simplifying the algorithm when compared to competitive approaches. We demonstrate the effectiveness of this method through case studies and metric evaluations that compare our proposed technique to those in the state-of-the-art.

2 Related Work

There are two main areas of research that are related to our proposed technique: shape improvement and Euler diagram drawing. In this section, we describe this related work and how these areas relate to each other.

2.1 Shape Improvement and Mean Curvature Flow

Mean curvature flow is a classic method for smoothing shapes. Each point on the curve (in two dimensions) or the surface (in three dimensions) is moved in the direction of the normal in a way that is proportional to its curvature. This process *flattens* the object [10]. Unfortunately, mean curvature flow also tends to *shrink* the object (in fact, differential geometers refer to this process as a *curve-shortening flow*), and this shortening may not be desired. Taubin's λ-μ smoothing is a classic method to avoid surface shrinking [28, 29]. The approach works by alternating the direction of improvement at every other step. Taubin shows that his proposed algorithm can smooth out rough parts of the model while preserving volume through an elegant analogy to Fourier

- *Paolo Simonetto and Carlos Scheidegger are with the University of Arizona. E-mail: paolosimonetto@email.arizona.edu and cscheid@cscheid.net*
- *Daniel Archambault is with Swansea University. E-mail: d.w.archambault@swansea.ac.uk*

Manuscript received 31 Mar. 2015; accepted 1 Aug. 2015; date of publication xx Aug. 2015; date of current version 25 Oct. 2015.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

space operations — the algorithm essentially reduces the amount of energy in high-frequency components while preserving the total amount of energy in the system. In this paper, we use the principle behind λ-μ smoothing to move vertices of the boundary along the direction of curvature. However, as set elements are located inside the diagram, we can use the repulsive forces of the elements for area preservation.

2.2 Euler Diagram Drawing

Methods for visualizing sets and their intersections has been an active area of research in recent years with many novel visualization approaches developed for this purpose [3]. As the algorithm we propose in this work improves diagrams that represent sets as polygons, we focus on these methods for set visualization in this section. However, it is important to note that other approaches exist for visualizing sets and their intersections that are based on circles [13, 26, 33], ellipses [16], and approaches that are not based on closed curves at all [1, 2, 15, 23].

A number of approaches are able to produce a drawing for every input instance. Rodgers *et al.* introduce a method for drawing a subclass of Euler diagrams [22] and then extend this research to work on any input instance [21]. In these approaches, the dual graph of the Euler diagram is transformed into a planar graph for drawing. Then, a triangulation is used to route set contours, creating a diagram. Simonetto *et al.* [25] propose a force-directed solution that is guaranteed to produce a correct drawing, if one exists. In this approach, a planar representation of the Euler diagram is computed and drawn initially with a planar graph drawing algorithm. Subsequently, the drawing is improved using a modification of the `PrEd` [6] algorithm to ensure that the structure of the Euler diagram does not change, nodes and edges do not cross boundaries during force-directed refinement, and that elements of the sets stay in their proper zones. This approach was further improved in `ImPrEd` [24], allowing for boundaries with adaptive complexities and faster convergence. Collins *et al.* [9] describe a method for drawing Euler diagrams when the elements of the sets have fixed positions. The approach relies on marching squares and implicit curves to derive the boundaries for each set. Riche and Dwyer [20] describe a method that prioritizes the containment of sets and splits sets in the diagram, when required, connecting them with edges. Stapleton *et al.* [27] present a method for inductively adding curves to a diagram and preserving well-formedness properties in the drawing. We demonstrate our technique operating on the output of many of these algorithms [9, 17, 20, 21, 25].

Dwyer *et al.* [11] propose a method for *topology-preserving constrained layout*, improving a variant of stress by alternated coordinate-wise gradient projection steps. This method is notable in that their stress variant is measured by the length of the *connector*, rather than the distance between vertex pairs. This allows polyline edges to be optimized, potentially shortening them. As we have already pointed out above that *"smoothing is curve shortening"*, this would appear to be a method for smoothing curves. However, the presence of *non-differentiable* shapes (from the node boxes) in their method means that in their setting, "jagged" polylines that route around node corners can be considered "smooth". Because our method uses forces as soft constraints, it tends to create shapes without sharp angles, which we argue are closer to our intuitive understanding of smoothness.

The closest work to our own is `eulerForce` [17] where boundaries and curve positions have been modified to improve the readability of the Euler diagram. This approach is based on a force system, consisting of fourteen forces, to optimize the shape of the curves in the diagram, driving them towards circular shapes.

In contrast, the approach presented in this paper is fundamentally different than this approach. Our force system is simpler, consisting of only four forces, which helps improve complexity in terms of execution speed and implementation. The results are more scalable and able to refine Euler diagrams of realistic complexity beyond the five or so sets of `eulerForce`. Finally, and most importantly, our approach can be configured to never produce an invalid drawing when it starts from a valid one – that is, we can impose constraints such that no new zones are created or destroyed by the refinement procedure.

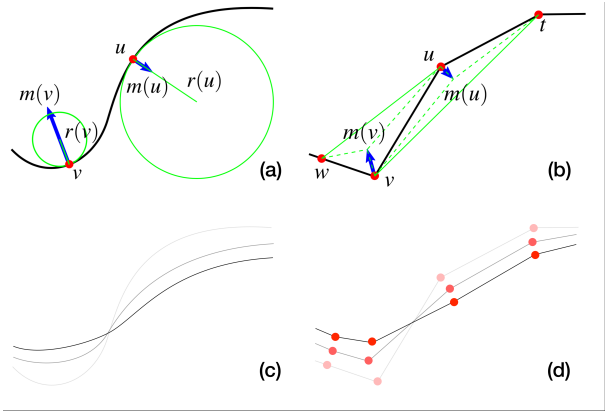

Fig. 2: Curve Shortening Flow. (a) In the original formulation of the problem, each point of the curve moves towards the centre of its osculating circle with speed proportional to the curvature at that point. (b) We adapt this problem to polylines by moving each point u of a boundary to the centroid of the triangle $\triangle tuv$, where t and v are the neighbour bends of u. (c) and (d) An evolution of the above curves after three iterations. The curves are smoothed from their initial to their final configuration (increasing saturation).

3 Method

In this section, we present a description of the proposed approach. Firstly, we describe how our simplified version of curve shortening flow can be adapted in order to optimize the smoothness of polygons. Then, we will analyse differences and similarities between the original and the adapted formulation of this method. Finally, we identify additional requirements needed in order to successfully apply this method to the optimisation of Euler diagrams.

3.1 Discrete Curve Shortening Flow

Curve shortening flow is defined on a continuous curve and in continuous time, ideally requiring the computation of infinite number of points for infinitesimal increments of time (see Fig. 2a). However, this algorithm operates in discrete time and space, as vertex positions can only be optimized on a discrete basis and the curves are modelled as polylines. Thus, we must adapt the original method to:

- *Discrete Space.* The concept of curvature, which is well defined for a sufficiently smooth curve, cannot be directly applied to a vertex u of a polygon $\ldots tuv \ldots$. The centre of curvature should be placed perpendicular to the line normal at u, but this line is not uniquely defined as the slope of the secant for a left and right increment from u only correspond when t, u, v are collinear.

- *Discrete Time.* In continuous curve shortening flow, the movement speed of a point can approach infinity when the radius of the osculating circle approaches zero. In continuous time, this problem is instantaneously corrected, as the speed of the vertex is re-calculated after an infinitesimally small period of time in the new lower stress position of the vertex. However, in discrete time, the time period t cannot be infinitesimally small, and therefore points with high speed will cause abrupt movements. Put simply, we require a more conservative approach to node movement in order to reduce the impact of large movements.

When defining our method for computing direction and intensity of movement, the properties of the original formulation must be preserved as much as possible. This formulation should be easy to implement and fast to compute. A solution that adequately satisfies all of these requirements computes the movement of a vertex u in $\ldots tuv \ldots$ along the vector that connects u to the centroid of the triangle $\triangle tuv$ (see Fig. 2b).

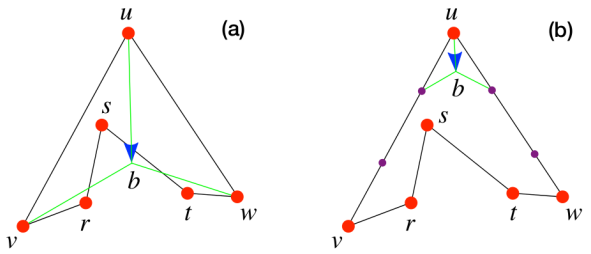

Fig. 3: Smoothing a concave polygon. (a) The smoothing operation of a vertex u would cause a self-intersection. (b) The chance of creating a self intersection is reduced by re-sampling the polygon to obtain smaller edges with uniform length. By adding the new bends (purple circles), the previous smoothing operation has no self intersections.

Definition 1. Given a polygon $p = p_0 \dots p_n$, we define *smoothing on* p_i as the operation that transforms p into $p' = p_0 p_{i-1} b p_{i+1} \dots p_n$ where b is the centroid of $\triangle p_{i-1} p_i p_{i+1}$.

Definition 2. Given a polygon $p = p_0 \dots p_n$, we define *smoothing on* p as the application of smoothing on each polygon vertex p_i.

3.2 Comparison with Curve Shortening Flow

We now describe properties of the centroid smoothing technique defined above and discuss similarities and differences between the definitions of discrete and continuous curve shortening flow.

Theorem 1. *Given a convex polygon p, by performing smoothing on any of its vertices, we obtain a polygon p' that is still convex.*

Proof. A polygon is convex iff all its internal angles are equal or less than 180°. Let $p = \dots tuv \dots$ and u be the vertex selected for smoothing. Since p is convex, the internal angles of t, u and v are equal or less than 180°. By performing smoothing on u, we substitute u with the centroid b of $\triangle tuv$. This operation does not increase the internal angle of vertices t and v, which will still be equal or less than 180°. Also, since the centroid of $\triangle tuv$ is contained in $\triangle tuv$, the angle $\angle tbv$ is not greater than 180°. Since all other internal angles remain unchanged, all the internal angles of the polygon are not greater than 180°, which means p' is still convex. □

Lemma 1. *Given a convex polygon p, by performing smoothing on any of its vertices, we obtain a polygon p' whose boundary do not self-intersect.*

Proof. As p is convex, p' is also convex and therefore simple. □

Theorem 2. *Each polygon p asymptotically converges to a single point by repeatedly applying smoothing on p.*

Proof. Let us consider a vertex u of a polygon $\dots tuv \dots$. If u is not collinear with t and v, by moving u to the centroid b of $\triangle tuv$, the polygon edge tu becomes tb and the polygon edge uv becomes bv, both of which are shorter than the original edges (or $\overline{tu} + \overline{uv} > \overline{tb} + \overline{bv}$). If u collinear with t and v, the edges computed after smoothing u will have a total length equal to the original edges ($\overline{tu} + \overline{uv} = \overline{tb} + \overline{bv}$). Since not all polygon vertices p_i can be collinear with the line defined by p_{i-1} and p_{i+1}, repeated application of smoothing on the polygon results in a sequence of polygons $p, p', p'' \dots$ of strictly decreasing perimeter. Since the polygon edges have positive lengths, the perimeter tends to zero or exactly one point. □

The previous theorems show that discrete curve shortening preserves the non-crossing properties for convex polygons, and that the asymptotic results of the two flows correspond for any polygon. However, the first property does not hold for concave polygons, as shown in Fig. 3a.

Algorithm 1 `EulerSmooth`.

```
for numberOfIterations do

    for all v ∈ V do
        force(v) ← 0
        constraint(v) ← +∞

                                          ▷ FORCE COMPUTATION
    for all ForceDef in ForceSystem do
        force ← force + ForceDef.compute()

                                          ▷ CONSTRAINT COMPUTATION
    for all ConstraintDef in ConstraintSystem do
        constraint ← min(constraint, ConstraintDef.compute())

                                          ▷ NODE MOVEMENT
    for all v ∈ V do
        if magnitude(force(u)) > constraint(u) then
            force(u) ← force(u) * constraint/magnitude(force(u))
        position(u) ← position(u) + force(u)

                                          ▷ POST-ITERATION
    for all PostIteration in PostIterationSteps do
        PostIteration.compute()
```

By re-sampling the polygon boundaries the chance of self intersection is reduced, as shown in Fig. 3b. The re-sampling procedure is sufficient to avoid self-intersections in all of our tested cases. However, we can enable an algorithm [24] used in previous approaches to provably avoid self-intersections. This algorithm checks for these intersections and reduces node movement to a safe distance. In practice, these checks are not required in most cases, causing an increase in computational complexity and running time. However, they can provide other benefits in a broad application scenario, motivating their inclusion in our algorithm (see Surrounding Edges in Section 4.3).

Boundary re-sampling provides a second, crucial advantage. When a polygon is over-sampled, the refinement of the boundaries is impeded. In fact, if a vertex u is very close to neighbours t and v, the triangle $\triangle tuv$ is small and therefore its centroid is close to u. Therefore, the re-sampling can reduce curve complexity, improving the speed of the computation.

3.3 Preservation of Diagram Properties

Discrete curve shortening flow provides a method for curve simplification. However, it cannot be used directly for Euler diagrams improvement without first ensuring that it will preserve fundamental diagram properties such as the presence or absence of set intersections and element containment. First, boundaries should not shrink to a point. As Euler diagrams contain set elements inside the curves, we use these elements as a backstop, allowing set boundaries to nicely shrink around the elements they contain. These elements prevent the diagram from collapsing to a single point. In section 4, we discuss this in further detail.

Secondly, sufficient spacing between set elements as well as between set elements and set boundaries needs to be ensured. Therefore, we propose to model discrete curve shortening flow using a force-directed algorithm, allowing us to use other forces to ensure proper spacing.

Finally, we must prevent set elements from crossing boundaries and ensure that set intersections in the diagram are neither created nor destroyed. This property is accomplished by constraining node movement using `ImPrEd` [24].

4 Implementation

In this section, we describe the implementation of a force-directed algorithm that can be used to improve the appearance of Euler diagrams

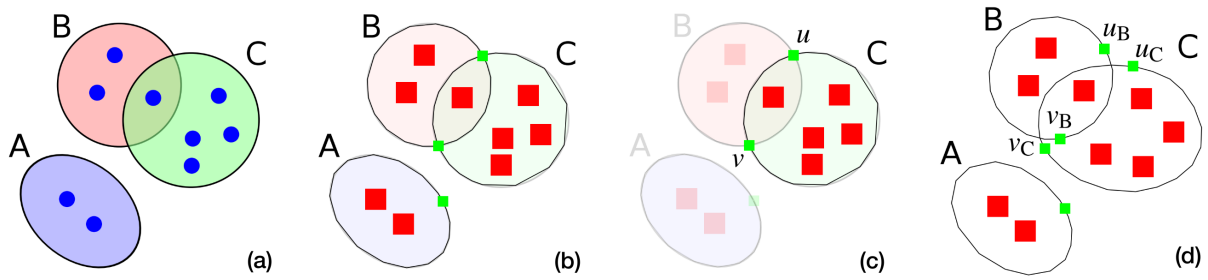

Fig. 4: The input diagram. (a) The original diagram the algorithm aims to improve. (b) The diagram is modelled as a graph. In green, the junctions. Polyline edges define the set boundaries. (c) For each set, we identify a subgraph containing all graph elements contained by that set. This subfigure shows the subgraph for set C. (d) We can choose to use independent boundaries, allowing for a less constrained refinement. Junctions and edges shared between subgraphs are duplicated (*e.g.* u is duplicated in u_B and u_C), allowing for separate refinement.

through discrete curve shortening flow. First, we explain how the input diagram, the output of any Euler diagram drawing method, is encoded into a graph. Then, we present the algorithm itself, `EulerSmooth`. Finally, we discuss how the algorithm can be configured to improve the diagrams presented in this paper.

4.1 Input Diagram

The input diagrams are modelled as graphs with polyline edges. These graphs have two sets of nodes: the set elements and the boundary junctions. The set elements have zero degree and have position, colour, size and labels. The junctions identify crossing points between boundaries of the original diagram. These nodes are connected with polyline edges that follow the boundary contours with arbitrary precision. Sets with no boundary crossings still have a singular junction that is placed on a random point of the set boundary, closing a polyline loop (see Figs. 4a and 4b). For each set, we identify a subgraph formed by all the elements contained in that set, and all junctions and edges that compose its boundary (see Fig. 4c).

This construction ensures that any crossings and concurrent boundaries are modified simultaneously over all sets, preserving the original topology of the graph. We define this condition as *dependent boundaries*. Whenever this behaviour is considered too restrictive, is it possible relax these constraints, duplicating junctions and shared edges. The duplicated elements share the same initial position but are free to evolve separately. We define this configuration as *independent boundaries* (see Fig. 4d). In this paper, all the input diagrams have been constructed by manually tracing over output images, constructing the input graph.

4.2 `EulerSmooth`

The proposed algorithm can be considered a modular version a force-directed approach. In the implementation (see Algorithm 1), forces, constraints, and post-iterations steps are modules that can be plugged into or removed from the algorithm.

The input corresponds to the Euler diagram represented as a graph composed of polyline edges. Polyline bends and segments are treated by most modules as if they were standard nodes and edges. However, we support polyline edges to differentiate elements of the drawing that can be added/removed (polyline bends and segments) versus those that must remain in the drawing (nodes and whole edges). All modules use a QuadTree data structure to accelerate the computation time.

We now discuss these modules, starting with the forces, then the constraints, and finally the post processing steps. Certain modules derive directly from `ImPrEd`, (EdgeContraction, NodeNodeRepulsion, SurroundingEdges, FlexibleEdges). Others have been substantially modified (EdgeNodeRepulsion). All remaining forces are new to `EulerSmooth`. Furthermore, the modular nature of the algorithm allows for greater flexibility than in `ImPrEd`. It is now possible, for example, to insert multiple instances of the same force as is done for EdgeNodeRepulsion in Section 4.3.

Most modules contain parameters that are fixed to constants. These constants were chosen empirically and based on common sense. They can be modified depending on the application. Further experiments should be run in order to determine appropriate values given a class of input or the desired properties of the output.

4.2.1 Forces

`EulerSmooth` allows flexibility in defining the force system to be used. In particular, for all the forces it is possible to indicate subsets of the graph elements to which the force is applied. Multiple instances of the same force can be loaded, allowing for different ideal distances to be enforced between different pairs of elements in the drawing.

In the following, we indicate with $\boldsymbol{p}_u \in \mathbb{R}^2$ the position of the node u. If $e = (u,v)$ is an edge, we denote the line segment between $\boldsymbol{p}_u$ and $\boldsymbol{p}_v$ as $\boldsymbol{p}_e$.

EdgeContraction(d) This force attracts the extremities of an edge toward each other, decreasing the length of the edge. The parameter d is the ideal edge length. The force $\mathscr{F}_u^c$ acting on node u by the edge $e = (u,v)$ is:

$$\mathscr{F}_u^c(e,d) = \left(\frac{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}{d}\right)(\boldsymbol{p}_v - \boldsymbol{p}_u)$$

NodeNodeRepulsion(d) This force repels two nodes from each other. The parameter d is the ideal distance between nodes. As is typical for force-directed algorithms, the edge contraction and extremities repulsion forces balance in d. The repulsive force $\mathscr{F}_u^r$ is:

$$\mathscr{F}_u^r(u,v,d) = \left(\frac{d}{\|\boldsymbol{p}_u - \boldsymbol{p}_v\|}\right)^2(\boldsymbol{p}_u - \boldsymbol{p}_v)$$

EdgeNodeRepulsion(d) This force repels nodes from nearby edges. The parameter d is the ideal distance between a node and a edge. This force has been modified from its original formulation in `ImPrEd` to improve its stability when nodes are not contained in the projection of an edge (Fig. 5). Let u be a node and $e = (v,w)$ be an non-incident edge ($u \notin e$). Let p be the projection of u onto the line defined by e. We define v as the closest edge extremity to u. The repulsive force $\mathscr{F}_u^e$ is:

$$\mathscr{F}_u^e(u,e,d) = \begin{cases} \left(\frac{d}{\|\boldsymbol{p}_u - \boldsymbol{p}_p\|}\right)^2(\boldsymbol{p}_u - \boldsymbol{p}_p) & \text{if } p \in \boldsymbol{p}_e \\ \mathscr{F}_u^r(u,v,d) & \text{otherwise} \end{cases}$$

The forces on the segment extremities are:

$$\mathscr{F}_v^e(u,e,d) = \begin{cases} -\mathscr{F}_u^e(u,e,d)\frac{\|\boldsymbol{p}_p - \boldsymbol{p}_w\|}{\|\boldsymbol{p}_v - \boldsymbol{p}_w\|} & \text{if } p \in \boldsymbol{p}_e \\ -\mathscr{F}_u^r(u,v,d) & \text{if } p \notin \boldsymbol{p}_e \end{cases}$$

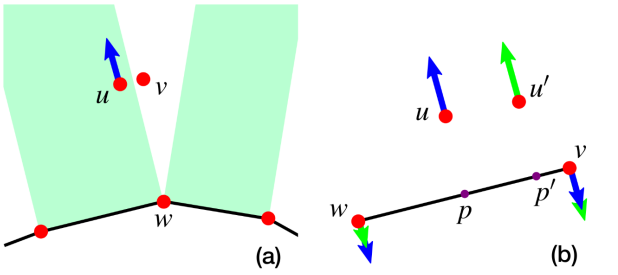

Fig. 5: Edge repulsion force modifications. (a) The related `ImPrEd` force is null if the node projection is not contained in the edge. As a result, a node might alternate between being affected or not by edge repulsion even with the slightest modification of position. The problem is solved by incorporating a repulsion force between the node and the closest edge extremity when the node projection is not contained in the edge. (b) The forces exerted on the edge extremities depend on their distance to p, providing a balancing effect on the forces.

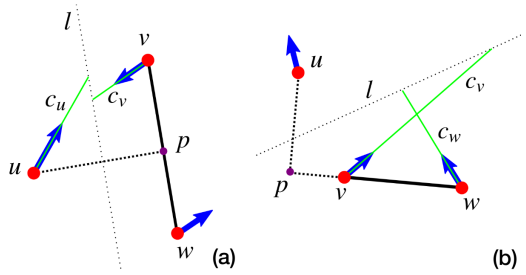

Fig. 6: Surrounding edge constraint. The module defines a line l that separates a node and and edge that should not be crossed. We ensure crossings will not occur by restricting the nodes movements to be smaller than the collision distance between c_x with l. When the force does not point towards l, no constraint is necessary. (a) Constraint computation for p when it lies on the edge. (b) Constraint computation for p when it lies outside the edge. Note that inverting the edge (v,w) is impossible when the edge forms part of a boundary as either v or w would need to cross an adjacent edge.

$$\mathscr{F}_w^e(u,e,d) = \begin{cases} -\mathscr{F}_u^e(u,e,d)\dfrac{\|\boldsymbol{p}_p - \boldsymbol{p}_v\|}{\|\boldsymbol{p}_v - \boldsymbol{p}_w\|} & \text{if } p \in \boldsymbol{p}_e \\ \mathbf{0} & \text{if } p \notin \boldsymbol{p}_e \end{cases}$$

The factor introduced in the first equation intensifies the repulsive force on the closest edge extremity and decreases it on the furthest one, better approximating the analogue physical system (see Fig. 5b).

CurveSmoothing This force applies the smoothing effect of the discrete curve shortening flow. The force moves a node toward the centroid of the triangle formed by its neighbours. Therefore, the force $\mathscr{F}_u^s$ that acts on a node u with neighbours t and v is:

$$\mathscr{F}_u^s = \frac{2}{3}\left(\frac{\boldsymbol{p}_t + \boldsymbol{p}_v}{2} - \boldsymbol{p}_u\right)$$

4.2.2 Constraints

These modules control node movements and/or avoid movements that could compromise diagram properties that we want to preserve. In this section, we describe these constraints.

DecreasingMaxMovement(d) This constraint gradually decreases the maximal movement a node can take in a time period, allowing for more precise final positioning. The initial value for this movement is set to d for all nodes and linearly decreases as the computation progresses.

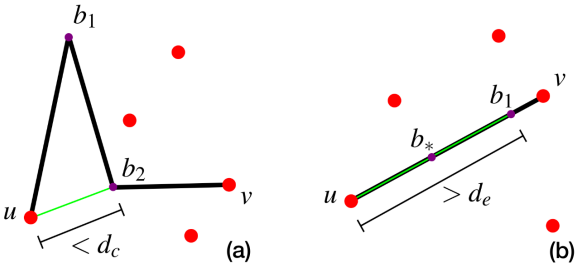

Fig. 7: Flexible edges contraction and expansion. (a) Contraction. Since b_1 previous bend (u) and following bend (b_2) are closer than d_c and no nodes are in the triangle $\triangle ub_1b_2$, the bend b_1 and its incident segments are substituted by the green segment. (b) Expansion. Since the first edge segment is longer than d_e, a new bend b_* is introduced and the segment is substituted by the two green ones.

MovementAcceleration(d) This constraint increases or decreases the speed of node movement according to consistency with the previous iteration, reducing oscillations. For a node v, the $c_i(u)$ is updated according to the angle a between consecutive iterations:

$$c_i(u) = \begin{cases} d & \text{if } i = 0 \\ c_{i-1}(1 + 2(1 - \frac{a}{60^\circ})) & \text{if } a < 60^\circ \\ c_{i-1} & \text{if } 60 \le a < 90^\circ \\ c_{i-1}/(1 + 4(\frac{a}{90^\circ} - 1)) & \text{if } a \ge 90^\circ \end{cases}$$

Thus, the constraint is multiplied by a factor in $[1,3]$ for movement in the same direction and divided by a factor in $[1,5]$ for movement in an opposite direction.

PinnedNodes This module constrains nodes to their input positions by setting their movement constraint to 0 throughout the computation.

SurroundingEdges This module constrains node movement so that nodes cannot cross edges denoted as surrounding edges. More specifically, given a set of nodes V and a set of edges E, this module ensures no $u \in V$ crosses any edge $e \in E$. For a node u and $e = (v,w)$, the module computes a line l that divides the plane into two halves. If the projection p of u lies in the edge ($p \in \boldsymbol{p}_e$), the line l is identified as the axis of symmetry for the segment up. If the projection lies outside the edge, l is the axis of symmetry of the segment uv, with v being the closest endpoint to u. This module ensures that u does not cross e by limiting movement to a distance between u and l where they cannot collide along the direction of movement (see Fig. 6).

4.2.3 Post-Iteration Steps

Post-iteration modules can process the the output of an iteration.

FlexibleEdges(d_c, d_e) This module re-samples the set boundaries. This re-sampling allows for set boundaries to grow and shrink, depending on the space required. Given an edge set E, the module increases or decreases the number of bends, depending on the stress of its edge segments. Given an edge e, the module first tries to remove bends or contract the edge, and then it tries to add bends or expand the edge. When contracting an edge, the module marks $b_1 \dots b_n$ for removal. The bend b_i is removed if the distance between its previous bend b_{i-1} (edge source, if $i = 1$) and the next bend b_{i+1} (edge target, if $i = n$) is less than a distance d_c and the triangle formed by the three bends is empty (see Fig. 7a). When expanding an edge, the segments $e_1 \dots e_n$ are checked for expansion. A segment is expanded when its length is greater than a distance d_e by inserting a new bend at its midpoint (see Fig. 7b).

4.3 Algorithm Configuration

In order to ensure comparable results, we run the same configuration of `EulerSmooth` for all the diagrams in the paper. We select a single distance parameter d^* and set three Boolean variables. The parameter

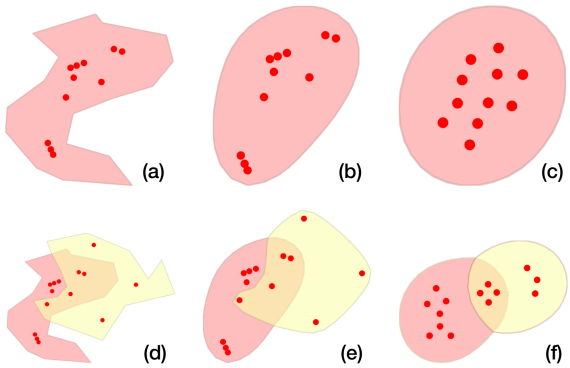

Fig. 8: Example diagrams optimised with `EulerSmooth`. (a,d) The original diagrams. (b,e) The diagrams optimised while keeping the nodes in their original position (`Fix`,`Dep`). (c,f) The diagrams optimised while allowing elements move (`Mov`,`Dep`).

d^* can be thought of as an ideal distance between set elements and between set elements and boundaries. This parameter is proportional to the scale of the imported diagram.

The core modules used for diagram improvement in this paper are:

CurveSmoothing
EdgeNodeRepulsion(d^*) between elements and boundary edges
EdgeContraction(0.7 d^*) for boundary edges
DecreasingMaxMovement(d^*)
MovementAcceleration(d^*)
FlexibleEdges(1.45 d^*, 1.5 d^*)

The first Boolean parameter sets whether dependent or independent boundaries are required (see Section 4.1). We denote `Dep` to indicate dependant boundaries and `Ind` to indicate independent boundaries:

if `Dep` **then**
    SurroundingEdges between all nodes and boundaries.
else
    SurroundingEdges between elements and boundaries.

Surrounding edges prevent set elements from crossing boundaries. For boundary nodes, we can relax this constraint, allowing for the creation or destruction of new zones not present in the input diagram. The result is generally a smoother diagram, where new empty zones are created. To our knowledge, no human centred experimentation exists that evaluates if these empty zones impede diagram readability. However, if this behaviour is not desired and the zones in the input diagram need to be preserved exactly, the surrounding edge constraint can be applied to all nodes in the drawing, including boundary nodes, ensuring that no new zone is created or destroyed during refinement.

The second Boolean parameter pins element movement. We use `Fix` to indicate the position of all set elements are fixed and `Mov` to indicate if they can move freely:

if `Fix` **then**
    PinnedNodes on set elements
else
    NodeNodeRepulsion(d^*) between set elements

Pinning elements is particularly useful when refining drawings where elements are specific locations, as in Bubble Sets [9]. Otherwise, the positions of set elements can be optimized for better distribution.

The final Boolean parameter allows concurrent boundaries to be relaxed. We use `Sep` to indicate the activation of this parameter:

if `Sep` **then**
    EdgeNodeRepulsion(d^* / 15) between boundary nodes and edges

When set, a local repulsive force is inserted between nodes and unrelated boundary edges, separating collinear boundaries. This option improves the readability of diagram but can generate stress and boundary irregularity on diagrams that do not contain concurrent boundaries.

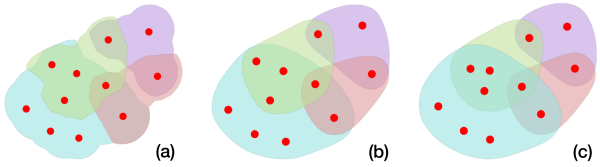

Fig. 9: Optimization of a `Set Visualiser` diagram [34]. (a) The input diagram. (b) The diagram optimised while keeping the nodes in their original position (`Fix`,`Ind`). (c) The diagram optimised while allowing elements move (`Mov`,`Ind`).

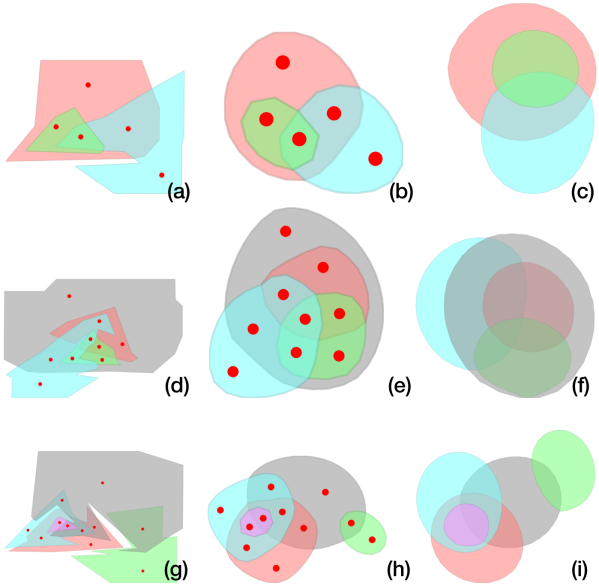

Fig. 10: Comparison of diagrams optimised with `EulerSmooth` and with `eulerForce`. In order to preserve the zone spacing with `EulerSmooth`, a dummy node have been inserted in each depicted zone. First row, example diagram with three sets (instance ID 1). Second row, example diagram with four sets (instance ID 1). Third row, example diagram with five sets (instance ID 2). (a,d,g) The input diagrams. (b,e,h) The diagrams optimised with `EulerSmooth` (`Mov`,`Dep`,`Sep`). (c,f,i) The diagrams optimised with `eulerForce`.

Fig. 10e shows how this option causes this separation between boundaries to form. Unnecessary tension is present top part of the turquoise set. Without separation enabled, the boundaries would almost overlap, similar to Fig. 10f, between the bottom of the green and grey sets.

For all images, we specify which Booleans are set to generate them.

Number of Iterations A final parameter is the number of iterations applied. A larger number of iterations is usually required for larger input. For all the examples, the number of iterations has been empirically determined. However, it is possible to modify the implementation to iterate until the user is satisfied or when a global graph improvement metric is reached.

5 Analysis

To evaluate the effectiveness of `EulerSmooth`, we extracted diagrams published in previous work and improved them with our technique. The initial diagrams have been created by tracing contours and adding set elements as prescribed by the output images. Then, these diagrams have been optimized using appropriate parameters for the given case. For example, when improving the Manhattan Bubble Sets diagram (see Fig. 13), we use fixed node positions as they correspond to locations on the map. Whenever applicable, we show the variety of outputs possible with `EulerSmooth`.

Contour Based Diagrams Euler diagrams drawing approaches might or might not show set elements, causing significant differences in how a diagram is created and interpreted. For example, in a diagram that does not represent set elements (contour based), the existence of a overlap in the diagram implies that the given intersection contains elements. On the other hand, this requirement is not imposed for element based approaches as the elements are either visible in the region or the region is empty.

For this reason, contour and element based approaches are not always directly comparable. In our case, since the algorithm assumes the diagram is element based and requires set elements to limit shrinking, we add dummy elements into the regions of contour-based approaches as shown in Fig. 10.

Testing Hardware and Software All images and results in this section are generated by a Java implementation of `EulerSmooth`, which includes a testing GUI that animates diagram smoothing, computes diagram statistics, and records running times. The source code of the application is available as supplimentary material[1].

All of our diagrams were created on a desktop machine equipped with an Intel Core i7-2600 processor, 8GB of RAM, and running KDE 4.14 on Arch Linux.

5.1 Qualitative Evaluation

Fig. 8 shows the results of `EulerSmooth` on two basic diagrams: one with a single set and a second with two overlapping sets. We notice how the contour regularity is improved regardless, but `EulerSmooth` achieves a better result when node movement is enabled.

Fig. 9 shows an improved version of a diagram generated with `Set Visualiser` [34]. By using independent boundaries, we can eliminate empty zones (blue-only region in the top-centre) which might make the original diagram less readable. We notice little difference between the results obtained with fixed versus movable elements as the initial positions of set elements do not require much refinement.

Fig. 10 shows the same input diagram improved with `EulerSmooth` and `eulerForce`. The diagrams improved with `eulerForce` appear more regular as our quantitative analysis in the next section can confirm. However, the shapes obtained with `EulerSmooth` are smooth and readable.

Fig. 11 shows the optimization of a diagram from Euler Representation [25, Figure 9a] consisting of twenty sets. This figure illustrates the difference between dependent and independent boundaries. In the first case, all regions of the original diagrams are preserved throughout the smoothing process, meaning that no new zones are created or destroyed. This forces concurrent boundaries to be preserved, limiting improvement. In particular, consider the blue set at the top of Fig. 11b: on the right side, the set crosses two other boundaries (green and red) at a single point. Since the point is required to satisfy the constraints of three different curves simultaneously, sharp angles are present in the drawing (blue and green).

If there is flexibility in preserving the exact topology of the input diagram, we enable independent boundaries and obtain the result in Fig. 11c. By enabling this option, we introduce new regions not present in input drawing (in Fig. 11c, the blue-only triangular region in the centre of the drawing). However, none of these new regions will contain elements and the boundaries are much easier to follow.

In Fig. 12, `EulerSmooth` is applied to an Untangled Euler diagram [20, Figure 1a]. The algorithm transforms it into a more classical looking diagram with regular boundaries.

Finally, Fig. 13 shows application of our method to the Manhattan Bubble Sets [9, Figure 9]. Although the shape of the sets in the final drawing are not circles, `EulerSmooth` can simplify boundaries even in these constrained circumstances, improving the readability of contours at the expense of additional occlusion of the background map.

5.2 Quantitative Evaluation

We evaluated the effectiveness of `EulerSmooth` by computing the average initial and final isoperimetric quotient for all set boundaries in the diagrams. Given a closed curve ℓ, the isoperimetric quotient computes the ratio between the area enclosed by the line and that of a circle with equal circumference: $Q = 4\pi \text{ area}(\ell) \,/\, \text{length}(\ell)^2$. Since a circle is the shape that maximizes the closed area for a given perimeter, all simple closed curves will have an isoperimetric quotient between 0 and 1. One can view the isoperimetric quotient as an indication of how close the closed curve is to a circle. We employ the average isoperimetric quotient to evaluate the quality of the contours as previous work reports that smooth shapes [5], and circles in particular [7], increase diagram readability.

Diagram	Fig.	Opt.	Iter.	Q_i %	Q_s %	Time (s)
General Euler	1b	MD	300	75.3	96.5	3.35
Untangled Small	1d	MI	100	36.7	76.3	2.42
Single	8b	FD	120	39.7	90.3	1.83
	8c	MD			99.6	1.96
Double	8e	FD	120	45.7	85.3	1.88
	8f	MD			98.5	2.07
Imdb20	—	FD	25	56.9	74.6	6.52
	—	FI	40		84.6	9.67
	11b	MD	25		73.9	6.71
	11c	MI	40		86.0	10.15
BubbleSet	13b	FI	50	4.5	10.2	6.43
Untangled	12b	FI	100	47.1	79.2	5.25
	12c	MI			82.8	5.21
SetVis	9b	FI	50	77.3	92.3	1.22
	9c	MI			93.3	1.19

Table 1: Statistics for the execution of `EulerSmooth` over the diagrams in this paper. The statistics for EulerForce diagrams (Figs. 10 and 14) are collected in Table 2. Constraints activated are indicated by their first letter (F for `Fix`, M for `Mov`, D for `Dep`, I for `Ind`). Iterations shows the number of smoothing cycles performed. Q_i is the initial average isoperimetric quotient. Q_s is the average isoperimetric quotient for the improved diagram. The running time is the average running time over five runs of the approach.

Improvements over Various Techniques Table 1 reports the enabled options, running time, and initial and final average isoperimetric quotient for the diagrams in this paper. We notice that our approach improves Q for all cases with larger improvements when less restrictive options are enabled (`Mov` instead of `Fix`, `Ind` instead of `Dep`). A low value is obtained for Bubble Sets (Fig. 13), as the diagram is heavily constrained and difficult to improve.

Running time and Complexity The reported running times are averaged over five executions and are in the range of one to twelve seconds. Times of this magnitude are acceptable for an interactive environment, as long as the data is only updated every few seconds. As expected, larger diagrams and independent boundaries are generally responsible for increased computation times. However, relatively simple diagrams might still require a high number of iterations. When observing the smoothing over time, we notice that boundary shapes are rapidly smoothed, but that a large amount of time is spent shrinking the contours to snugly fit the set elements. In fact, the shrinking process is slower on regular shapes.

Comparison with `eulerForce` We run a direct comparison between `EulerSmooth` and `eulerForce`. The authors of `eulerForce` provide a software that can generate an initial random diagram consisting of three, four, or five sets. We generate five diagrams for each (a total of 15 diagrams) and optimize them using `EulerSmooth` and `eulerForce`. We compare the resulting average isoperimetric quotient and record the average running time for both algorithms. Table 2 reports this data.

The computed metrics indicate a higher set boundary regularity for `eulerForce`, as suggested by visual inspection. The values are high and about the same for both `EulerSmooth` and

[1] `http://hdc-arizona.github.io/EulerSmooth/`

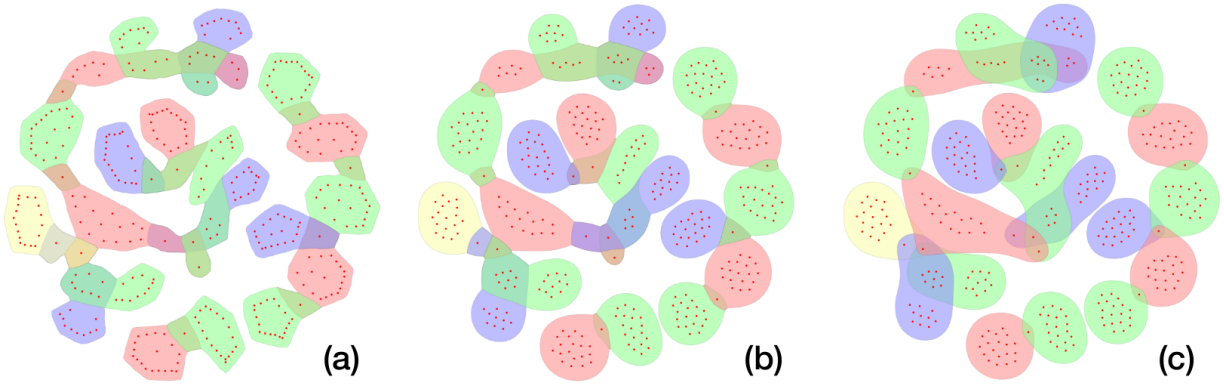

Fig. 11: Optimization of the Imdb20 Euler diagram [25, Figure 9a]. (a) The input diagram. (b) The diagram optimised while allowing elements to move (`Mov`,`Dep`). (f) The diagram optimised allowing elements move and sets evolve independently (`Mov`,`Ind`).

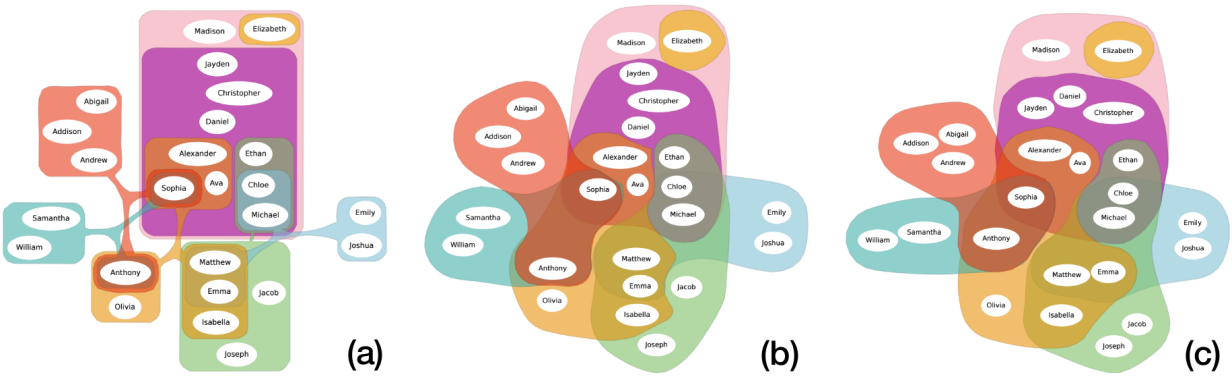

Fig. 12: Optimization of an Untangled Euler Diagram [20, Figure 1a]. (a) The input diagram. (b) The diagram optimised while pinning elements in their input positions (`Fix`,`Ind`). (c) The diagram optimised while allowing elements to move (`Mov`,`Ind`).

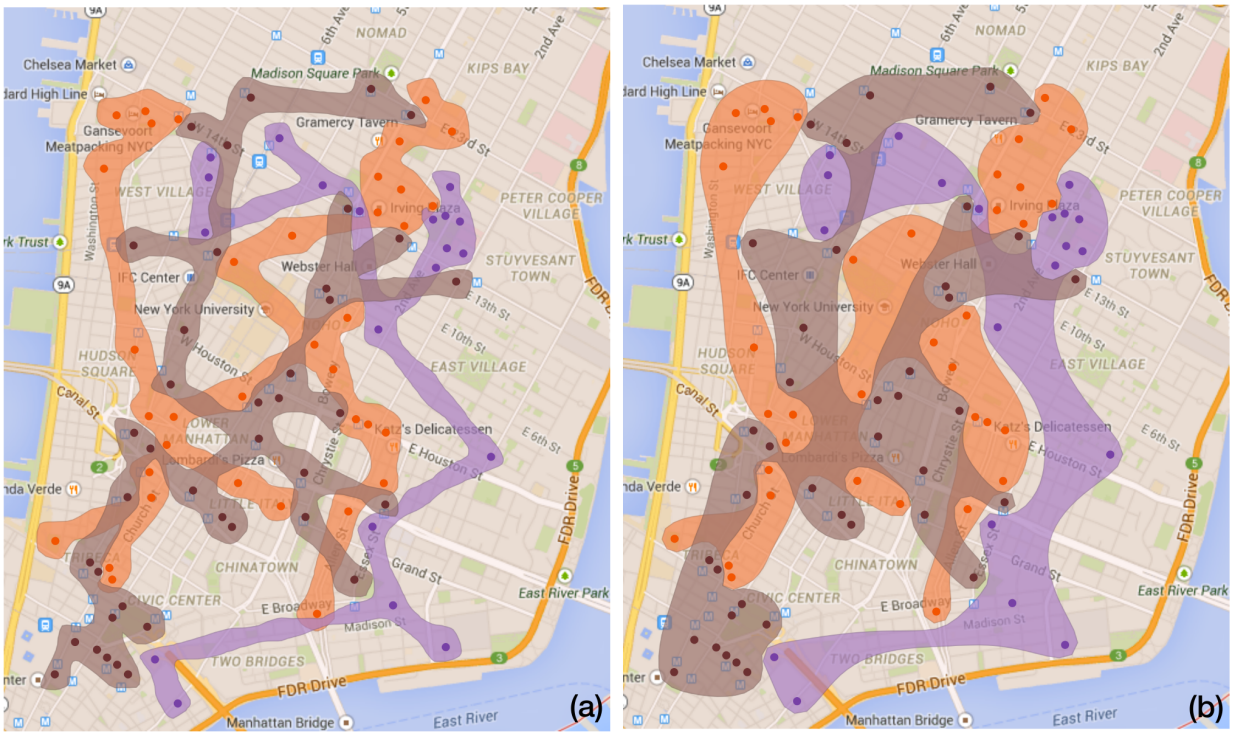

Fig. 13: Improvement of the Bubble Sets diagram over the Manhattan map [9, Figure 9]. (a) The input diagram. (b) The diagram optimised with `EulerSmooth` (`Fix`,`Ind`).

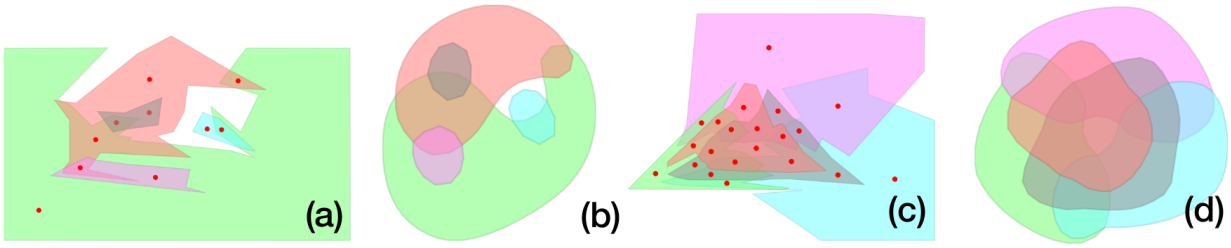

Fig. 14: Optimization of two five-set diagrams unsupported by `eulerForce`. (a,c) The input diagrams. (b,d) The diagrams optimised with `EulerSmooth` (`Mov`,`Dep`,`Sep`). The dummy elements have been removed to create a contour based diagram.

Sets	Id	Q_i %	Q_s %	Q_f %	Time$_s$ (s)	Time$_f$ (s)
3	1	58.3	95.5	99.0	3.6	2.35
	2	57.0	95.6	98.5	3.5	3.48
	3	51.3	96.1	98.6	3.6	2.25
	4	51.0	96.2	97.4	4.6	2.57
	5	58.5	95.5	98.8	5.9	2.57
4	1	53.6	96.2	98.1	5.4	4.85
	2	38.4	93.9	94.0	5.7	5.85
	3	45.5	94.9	97.2	6.2	4.63
	4	41.9	93.3	93.0	4.5	6.14
	5	53.6	95.6	98.5	5.1	5.85
5	1	36.5	84.9	Invalid	8.6	32.89
	2	41.7	95.2	98.2	8.2	16.64
	3	43.0	95.6	95.0	8.2	14.53
	4	44.3	86.2	Invalid	13.9	25.67
	5	42.6	85.6	Invalid	13.5	22.47

Table 2: Statistics for the improvement of diagrams with `EulerSmooth` and `eulerForce`. Q_x is the average isoperimetric quotient. Subscript i indicates initial, s indicates `EulerSmooth`, f indicates `eulerForce`. Q_f has not been computed for the three 5-sets instances (ID 1, 4 and 5) as `eulerForce` produced invalid drawings. On all diagrams, we run `EulerSmooth` with the option `Mov`, `Dep` and `Sep`. For 3 and 4 sets, 350 iterations were sufficient. For 5 sets, we ran 500 iterations.

Sets	Fig.		1.2^{-2} %	1.2^{-1} %	1.2^{1} %	1.2^{2} %
3	10a	Time	181	126	73	57
		Q_s	103	102	97	89
4	10d	Time	188	137	77	57
		Q_s	99	100	99	97
5	10g	Time	223	147	70	52
		Q_s	101	101	96	87

Table 3: Scalability of the approach in terms of the change in running time and isoperimetric quotient. We tested five levels of sampling where the number of points on the boundary is multiplied by factors of 20% (1.2^i for $i \in \{-2,-1,0,1,2\}$). The first two factors increase whereas the last two factors reduce the number of samples with respect to the original input diagram. All changes are normalized relative to the original input diagram, and all levels are run for a total of 350 iterations. This data quantifies running time and drawing quality trade-offs for finer and coarser samplings of the boundary.

`eulerForce`, indicating that all output diagrams are of high quality. However, `eulerForce` does not always produce a valid diagram. `eulerForce` produces invalid diagrams (zones are created or destroyed during execution) for three out of five instances of the five set data set. The authors explain that an increase in diagram complexity increases the probability of invalid diagrams, resulting in a success rate of 61% for diagrams of five sets. As seen in this paper, we can scale to diagrams of twenty sets.

`EulerSmooth`, due to its simple force system, can generate correct and pleasant diagrams even for the instances not supported by `eulerForce` (see Fig. 14). Also, the running time of `EulerSmooth` is faster than `eulerForce` for diagrams of four to five sets, indicating that it has increased scalability.

Scalability of `EulerSmooth` An algorithmic complexity for `EulerSmooth` is difficult to formalize as the number of boundary nodes, and thus the number of nodes in the diagram, are constantly changed by re-sampling. Typically, the computation time per iteration decreases as the algorithm progresses, as boundaries are usually simplified by the approach and fewer points are necessary to define them. When the algorithm converges, often the boundaries are as simple as possible.

To quantify the scalability of our approach, along with the trade-off between drawing quality and running time, we test a number of sampling levels for three Euler diagrams. Table 3 reports these results. These results are obtained by first increasing and then decreasing the number of points on the boundaries. Finer boundary sampling leads to a much higher running time with a slight increase in drawing quality. On the other hand, coarser boundary sampling drastically reduces running time at the expense of diagram quality.

In order to provide a fair comparison between all levels, we run 350 iterations for each level. This number of iterations is not always sufficient to reach a stable drawing for boundaries with a finer sampling and is more than enough for boundaries with a coarser sampling. Therefore, in the first case, we can run more iterations of the algorithm to increase diagram quality. Conversely, we can run fewer iterations in the second case to further improve the running time. For these reasons, the level of sampling can be used to mitigate scalability issues. In particular, when the standard sampling level leads to excessive running times in large diagrams.

6 Conclusion

In this paper, we present a simple approach for Euler diagram improvement based on a simplification of curve shortening flow. We translate this method into a force system that can be used to improve the output of any Euler diagram drawing method using polygonal curves and set elements. The approach has been demonstrated on a variety of methods [9, 17, 20, 21, 25] for drawing Euler diagrams and evaluated against competitive approaches.

As future work, it would be interesting to see if we can extend other aspects of curve shortening flow and vector field design to the improvement of Euler diagrams in general. As methods exist that do not use a force system to optimize the input shape, it may be beneficial, in terms of computation time, to use these methods instead of a force system to optimize Euler diagrams.

Furthermore, an area of future work would be to provide evidence of effectiveness of this refinement through human centred experimentation. Although curves with high isoperimetric quotients often correspond to more readable curves, our contour optimization technique can affect other aesthetic criteria in the diagram, influencing the readability of the drawing.

REFERENCES

[1] B. Alper, N. H. Riche, G. Ramos, and M. Czerwinski. Design study of linesets, a novel set visualization technique. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '11)*, 17(12):2259–2267, 2011.

[2] B. Alsallakh, W. Aigner, S. Miksch, and H. Hauser. Radial sets: Interactive visual analysis of large overlapping sets. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '13)*, 19(12):2496–2505, 2013.

[3] B. Alsallakh, L. Micallef, W. Aigner, H. Hauser, S. Miksch, and P. Rodgers. Visualizing sets and set-typed data: State-of-the-art and future challenges. In *Proc. of Eurographics Conference on Visualization (EuroVis '14) STAR Reports*, pages 1–21, 2014.

[4] G. D. Bader and C. W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(2):27, 2003.

[5] F. Benoy and P. Rodgers. Evaluating the comprehension of Euler diagrams. In E. Banissi, R. A. Burkhard, G. Grinstein, et al., editors, *International Conference on Information Visualisation (IV07)*, pages 771–780. IEEE Computer Society, 2007.

[6] F. Bertault. A force-directed algorithm that preserves edge crossing properties. *Information Processing Letters*, 74(1–2):7–13, Apr. 2000.

[7] A. Blake, G. Stapleton, P. Rodgers, L. Cheek, and J. Howse. The impact of shape on the perception of Euler diagrams. In *Proc. of the 8th International Conference on the Theory and Application of Diagrams*, volume 8578 of *LNAI*, pages 124–138, 2014.

[8] G. Chen, K. Mischaikow, R. S. Laramee, P. Pilarczyk, and E. Zhang. Vector field editing and periodic orbit extraction using morse decomposition. *Visualization and Computer Graphics, IEEE Transactions on*, 13(4):769–785, 2007.

[9] C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '09)*, 15(6):1009–1016, 2009.

[10] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324. ACM Press/Addison-Wesley Publishing Co., 1999.

[11] T. Dwyer, K. Marriott, and M. Wybrow. Topology preserving constrained graph layout. In I. G. Tollis and M. Patrignani, editors, *International Symposium on Graph Drawing (GD08)*, volume 5417 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2009.

[12] L. Euler. Lettres à une princesse d'Allemagne. *letters no. 102-108*, 1761.

[13] H. A. Kestler, A. Mller, J. M. Kraus, M. Buchholz, T. M. Gress, H. Liu, D. W. Kane, B. R. Zeeberg, and J. N. Weinstein. VennMaster: Area-proportional Euler diagrams for functional go analysis of microarrays. *BMC Bioinformatics*, 9(1), 2008.

[14] K. Koffka. *Principles of Gestalt Psychology*. Harcourt Brace, 1935.

[15] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister. UpSet: Visualization of intersecting sets. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):1983–1992, 2014.

[16] L. Micallef and P. Rodgers. eulerAPE: Drawing area-proportional 3-Venn diagrams using ellipses. *PLoS One*, 9(7):101717, 2014.

[17] L. Micallef and P. Rodgers. eulerForce: Force-directed layout for Euler diagrams. *Journal of Visual Languages and Computing*, 25(6):924–934, 2014.

[18] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[19] S. E. Palmer. Common region: a new principle of perceptual grouping. *Cognitive Psychology*, 24(3):436–447, 1992.

[20] N. H. Riche and T. Dwyer. Untangling Euler diagrams. *IEEE Trans. on Visualization and Computer Graphics (InfoVis '10)*, 16(6):1090–1099, 2010.

[21] P. Rodgers, L. Zhang, and A. Fish. General Euler diagram generation. In *Diagrammatic Representation and Inference*, volume 5223 of *LNCS*, pages 13–27. Springer, 2008.

[22] P. Rodgers, L. Zhang, G. Stapleton, and A. Fish. Embedding wellformed Euler diagrams. In *Proc. of the 12th International Conference on Information Visualisation*, pages 585–593, 2008.

[23] R. Sadana, T. Major, A. Dove, and J. Stasko. OnSet: Visualizing Boolean set-typed data using direct manipulation. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):1993–2002, 2014.

[24] P. Simonetto, D. Archambault, D. Auber, and R. Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum (EuroVis '11)*, 30(3):1071–1080, 2011.

[25] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum (EuroVis '09)*, 28(3):967–974, 2009.

[26] G. Stapleton, J. Flower, P. Rodgers, and J. Howse. Automatically drawing Euler diagrams with circles. *Journal of Visual Languages & Computing*, 23(3):163–193, 2012.

[27] G. Stapleton, P. Rodgers, J. Howse, and L. Zhang. Inductively generating Euler diagrams. *IEEE Trans. on Visualization and Computer Graphics*, 17(1):88–100, 2011.

[28] G. Taubin. Curve and surface smoothing without shrinkage. In *Proc. of the Fifth International Conference on Computer Vision*, pages 852–857, 1995.

[29] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM, 1995.

[30] A. Treisman and J. Souther. Search asymmetry: a diagnostic for preattentive processing of separable features. *Journal of Experimental Psychology: General*, 114(3):285–310, 1985.

[31] A. Verroust and M.-L. Viaud. Ensuring the drawability of extended Euler diagrams for up to 8 sets. In *Diagrammatic Representation and Inference*, volume 2980 of *LNCS*, pages 128–141. Springer, 2004.

[32] K. Wade, D. Greene, C. Lee, D. Archambault, and P. Cunningham. Identifying representative textual sources in blog networks. In *Proc. of the International Conference on Weblogs and Social Media (AAAI ICWSM)*, pages 393–400, 2011.

[33] L. Wilkinson. Exact and approximate area-proportional circular Venn and Euler diagrams. *IEEE Trans. on Visualization and Computer Graphics*, 18(2):321–331, 2012.

[34] D. Wyatt, D. Wynn, and P. Clarkson. Set visualiser, 2009. `https://www-edc.eng.cam.ac.uk/tools/set_visualiser/`.

[35] C.-Y. Yao, M.-T. Chi, T.-Y. Lee, and T. Ju. Region-based line field design using harmonic functions. *Visualization and Computer Graphics, IEEE Transactions on*, 18(6):902–913, 2012.