Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 19, no. 2, pp. 571–593 (2015)
DOI: 10.7155/jgaa.00364

MapSets: Visualizing Embedded and Clustered Graphs

Alon Efrat[1] *Yifan Hu*[2] *Stephen G. Kobourov*[1] *Sergey Pupyrev*[1,3]

[1]Department of Computer Science
University of Arizona, Tucson, USA
[2]Yahoo Labs, New York, USA
[3]Institute of Mathematics and Computer Science
Ural Federal University, Ekaterinburg, Russia

Abstract

In addition to objects and relationships between them, groups or clusters of objects are an essential part of many real-world datasets: party affiliation in political networks, types of living organisms in the tree of life, movie genres in the internet movie database. In recent visualization methods, such group information is conveyed by explicit regions that enclose related elements. However, when in addition to fixed cluster membership, the input elements also have fixed positions in space (e.g., georeferenced data), it becomes difficult to produce readable visualizations. In such fixed-clustering and fixed-embedding settings, some methods produce fragmented regions, while other produce contiguous (connected) regions that may contain overlaps even if the input clusters are disjoint.

Both fragmented regions and unnecessary overlaps have a detrimental effect on the interpretation of the drawing. With this in mind, we propose MapSets: a visualization technique that combines the advantages of both methods, producing maps with non-fragmented and non-overlapping regions. The proposed method relies on a theoretically sound geometric algorithm which guarantees contiguity and disjointness of the regions, and also optimizes the convexity of the regions. A fully functional implementation is available in an online system and is used in a comparison with related earlier methods.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
November 2014	May 2015	June 2015	August 2015	August 2015

Published: November 2015

Article type:	Communicated by:
Regular Paper	C. Duncan and A. Symvonis

A preliminary version of this paper was presented at the *22nd International Symposium on Graph Drawing* (GD 2014) [16]

E-mail addresses: alon@cs.arizona.edu (Alon Efrat) yifanhu@yahoo.com (Yifan Hu) kobourov@cs.arizona.edu (Stephen G. Kobourov) spupyrev@gmail.com (Sergey Pupyrev)

1 Introduction

In many real-world examples of relational datasets, groups of objects (clusters) are an inherent and important part of the input. For example, scientists belong to specific research communities, politicians are affiliated with specific parties, and living organisms are divided into biological species in the tree of life. The clusters are often visualized with regions in the plane that enclose related objects. Such representations are very helpful in dealing with clustered data. By explicitly defining the boundary and coloring of the regions, the cluster information becomes evident. In many instances the data objects are often associated with fixed or relative positions in the plane. In geo-referenced data, for example, the positions of the objects might be based on their geographic coordinates. Thus a natural problem arises: *How to best visualize graphs in which vertices are divided into clusters and embedded with fixed positions in the plane?*

Several existing visualization approaches seem suitable. For example, methods for visualizing set relations over existing embedded pointsets, such as BubbleSets [11] and LineSets [3] use colored shapes to connect objects that belong to the same set. Alternatively, a geographic map metaphor can be used to represent such data. With self-organizing maps [29] or geometry-based GMaps [23], objects become cities and cluster information is captured by uniquely colored countries. While both approaches can produce compelling visualizations, we argue that neither is perfectly suited to the problem of visualizing embedded and clustered graphs.

As the number of sets increases, set-based methods generate very complex and sometimes ambiguous results. More recent methods such as KelpDiagrams [12] and KelpFusion [32] reduce visual clutter and guarantee unambiguous visualization. But more importantly, all of these methods result in overlapping regions for the sets, even when the input sets are disjoint. This unnecessarily increases visual complexity and might mislead the viewer about the disjointness of the sets. The geographic map approach suffers from a different problem. A country in the map, that represents a given cluster of vertices, might not be a contiguous (connected) region in the plane. Even though each cluster is uniquely colored, such fragmented maps are difficult to read as human perception of color changes based on surrounding colors [37] and can be misinterpreted [26].

We want to combine the advantages of existing methods, while attempting to avoid their problems. That is, we are interested in visualizing embedded and clustered graphs with non-fragmented and non-overlapping regions. While constructing such representations is easy in theory, in practice the regions may still have high visual complexity; see Figure 1. Ideally the regions should be as *convex* as possible, as the convex hull best captures cohesive grouping according to Gestalt theory [27]. Riche and Dwyer [38] simplify Euler diagrams, relying on a similar argument that the use of convex and simple regions is a primary factor impacting readability.

With this in mind, we describe MapSets, a method for creating non-fragmented, non-overlapping regions that are as convex as possible, from a given embedded and clustered graph. We consider several criteria

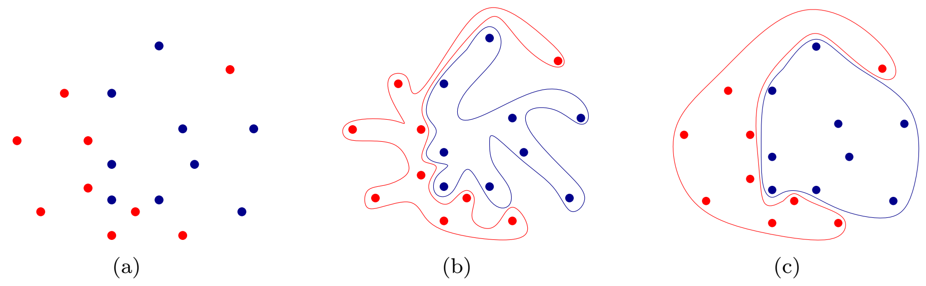


Figure 1: (a) An embedded and clustered (red/blue) pointset. (b-c) Two different ways to construct contiguous shapes bounding points of the same color.

for measuring convexity of a given shape, and propose a novel geometric problem aiming at optimizing convexity. We present a theoretical analysis of the problem in Section 3, which includes an observation of its computational hardness and a new approximation algorithm. Next, in Section 4, we provide a practical method for visualizing clustered graphs, which relies on the theoretical algorithm and guarantees contiguity and disjointness of the regions, while also optimizing the convexity of the regions. A fully functional implementation is available in an online system and is used in a comparison with existing techniques, which we present in Section 5.

2 Related Work

We review work related to the practical and theoretical aspects of the problem of visualizing embedded and clustered graphs.

2.1 Set Visualization

Graph clusters can be viewed as sets over graph vertices. In Venn diagrams [9,15] and their generalization, Euler diagrams, closed curves correspond to (possibly overlapping) sets, and overlaps between the curves indicate intersections. Several techniques have been recently devised to automatically generate Euler-like diagrams. Simonetto et al. [40] and Stapleton et al. [43] automatically generate Euler-like diagrams, by allowing disconnected regions, which can be complex and non-convex, especially when the sets exhibit numerous overlaps. Riche and Dwyer [38] propose a way to avoid the visual complexity problem by drawing simplified rectangular Euler-like diagrams, that do not depict the intersections between the sets explicitly, by creating separate rectangular regions for the sets, and duplicating objects that belong to multiple sets. In a user study, they found that it is beneficial to show intersections using simple set regions and strict containment, enabled by the duplication.

For the setting where the positions of the objects are fixed, Collins et al. [11] present BubbleSets, a method based on isocontours to overlay such an arrangement with enclosing set regions. A similar approach is suggested by Byelas and Telea [7]. The readability of these visualizations suffer when there are many overlapping regions.

A recent technique called LineSets [3] aims to improve the readability of complex set intersections and to minimize the overall visual clutter by reducing set regions to simple curved lines drawn through set elements. KelpDiagrams [12] incorporate classic graph-drawing "bubble and stick" style graph or tree spanners over the member points in a set. When multiple sets are shown simultaneously, overlapping regions are drawn with strictly nested containment. KelpFusion [32] adds filled-in regions to provide a stronger sense of grouping for close elements. A significant limitation of all these set visualization techniques is that they produce overlapping regions even when the sets are disjoint.

2.2 Visualizing Graphs as Maps

The geographic map metaphor is utilized as visual interface for relational data, where objects, relations between objects, and clustering are captured by cities, roads, and countries. Using maps to visualize non-cartographic data has been considered in the context of spatialization by Fabrikant et al. [17]. Self-organizing maps, coupled with geographic information systems, render 2D maps of textual documents [41], which provide an adaptable set of tools for spatial visualization of large document collections. Maps of science showing groups of scientific disciplines are used by a wide range of professionals to grasp developments in science and technology [6]. One drawback is that self-organizing maps are very computationally expensive.

The geographic map metaphor is helpful for graph and network visualization. Gronemann and Jünger [21] show how to draw clustered graphs exploiting elevation levels to show the cluster hierarchy. An alternative Graph-to-Map (GMap) approach [23] combines graph layout and graph clustering, together with appropriate coloring of the clusters and creating boundaries based on clusters and connectivity in the original graph. Maps of computer science [19], built on top of GMap, provide a way for visual exploration of topics in a particular conference or journal. However, since layout and clustering are two separate steps, a region representing a cluster may often be fragmented; see Figure 7(b). Such fragmentation makes it difficult to identify the correct regions and can result in misinterpretation of the map [26]. Note that in the setting when either an input embedding or clustering can be modified, the GMap approach can be improved to achieve contiguous regions [28], which results in better performance for cluster-based tasks [39].

2.3 Colored Spanning Trees

From an algorithmic perspective, our approach of optimizing the convexity of regions that cover points in the plane is related to several computational geometry problems; we mention only a few of them here. In many problems the input is a multicolored point set, as in the red-blue intersection, separation, and connection problems [1, 4, 5]. Tokunaga [44] considers a set of bicolored points in the plane and computes one geometric spanning tree of each color such that they intersect as few times as possible. A related work is the group Steiner tree problem where, for a graph with colored vertices, the objective is to find a minimum weight subtree covering all colors [33]. Also related is the problem of computing spanning graphs for multicolored point set of Ferran et al. [24]. The problem is motivated by optimizing the amount of "ink" needed to connect monochromatic points that arise when visualizing sets using KelpFusion. Note that these trees cannot be directly used as "skeletons" of regions in the plane as they can result in overlapping regions.

Finally, the clustered planarity problem is related to our approach [18]. The problem asks for a drawing of a clustered graph such that every cluster is represented by a non-fragmented region containing exactly the vertices in the cluster. Unlike our setting, the graph edges are taken into account; the drawing must be planar and edges of the graph can cross region boundaries at most once. While many special cases have polynomial-time algorithms, the computational complexity of the general variant remains open.

3 Creating Contiguous Non-Overlapping Regions

We assume that the input instance consists of a finite set of objects P with fixed positions $p_i \in \mathbb{R}^2$ for all $i \in P$, for example, cities and their geographic locations. In practical applications labels are often associated with the objects. In this case, we assume that non-overlapping bounding boxes for the labels are given. The input also specifies a clustering $C = \{C_1, \ldots, C_k\}$ of the objects with $\cup_{i=1}^{k} C_i = P$ and $C_i \cap C_j = \emptyset$ for $i \neq j$. We wish to enclose all objects of the same cluster by a single contiguous region so that regions corresponding to different clusters do not overlap.

On the one hand, simply overlaying each cluster with a convex region (e.g., bounding box or convex hull) is not always a valid solution, as it might cover elements in other clusters. On the other hand, representing clusters by some minimal regions (e.g., spanning or Steiner trees) is also not always valid, as it might result in intersecting regions.

We require regions that are contiguous and disjoint, and it is not difficult to see that such regions can be easily computed. We can begin by computing a crossing-free spanning tree of points belonging to some cluster. Once the tree is constructed, its vertices and edges become "obstacles" that should be avoided by subsequent trees. Note that we can process all the clusters in this manner, as the trees do not separate the plane into more than one region. Finally, contiguous non-overlapping regions can be grown, starting from these disjoint trees. However, this procedure often generates "octopus"-like shapes that are neither aesthetically pleasant nor practically useful for visualization; see Figure 1. Hence, we require a method for creating regions that are as

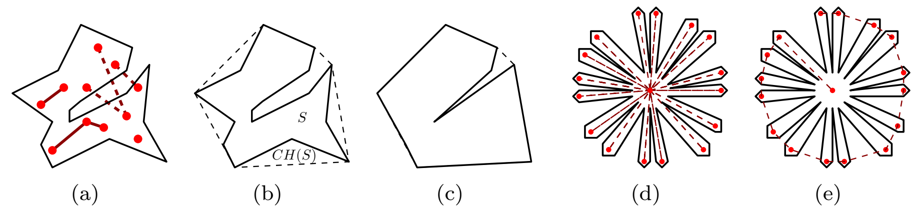


Figure 2: Convexity measures for a shape S enclosing red points. (a) Solid segments are within S, while dashed ones are not. (b) A shape and its convex hull (dashed). (c) Area-based measure ignores boundary defects. (d-e) Ink needed to connect the points is much bigger than the length of the minimum spanning tree. The shape is enclosed in solid black, while the tree is dashed red.

convex as possible. In order to design such a method, a quality criterion for measuring the convexity of regions is needed. Next we review and formalize several convexity measures.

3.1 Convexity Measures

A shape S is said to be convex if it has the following property: If points $p, q \in \mathbb{R}^2$ belong to S then all points along the line segment $\overline{pq}$ belong to S as well. The definition allows for several different ways to measure the convexity of non-convex shapes.

Point/vertex visibility. For a given shape S, this convexity measure is defined as the probability that for points p and q, chosen uniformly at random from S, all points from the line segment $\overline{pq}$ also belong to S [46]. The result is a real number in the range $[0, 1]$, with 1 corresponding to convex shapes.

A problem with this definition is that it is difficult to compute, even if S is a polygon. Therefore, we suggest its discrete variant, taking into account that the input of our problem specifies points in the plane; see Figure 2(a).

This measure takes into account how many segments $\overline{pq}$ are completely in S for pairs of input points $p, q \in P$ of the cluster corresponding to S. The measure is defined as $\frac{\sum_{p,q \in P} \delta(p,q)}{|P|^2}$, where the sum is over all pairs of input points P and $\delta(p, q) = 1$ if $\overline{pq}$ lies inside S and $\delta(p, q) = 0$, otherwise.

Convex hull area/perimeter. Recall that the smallest convex set which includes a shape S is called the *convex hull*, $CH(S)$, of S; see Figure 2(b). The area-based convexity measure is defined as $\frac{Area(S)}{Area(CH(S))}$; it is frequently used and appears in textbooks [42]. The result is a real number in the range $[0, 1]$, with 1 corresponding to convex shapes. Unlike visibility-based measures, the convex hull-based one is very easy to calculate efficiently and is robust with respect to noise. However, the definition does not allow to detect large anomalies of the

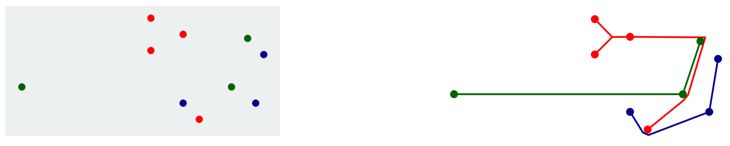

(a) (b)

Figure 3: (a) An input for CESF with $n = 10$ points and $k = 3$ colors. (b) An optimal solution with minimum ink containing Steiner points.

boundary that have a relatively small impact on the shape area; see Figure 2(c). The perimeter-based definition attempts to remedy this: $\frac{Perimeter(S)}{Perimeter(CH(S))}$.

If a shape S is convex, then every spanning tree on the given point set lies completely in S. On the other hand, non-convex shapes do not necessarily admit such a spanning tree. Hence, the length of a shortest curve that belongs to S and connects all the input points is an indicator of convexity of S. In the following measure, we compare the length of such a curve (or equivalently, the amount of "ink" needed to connect all the points) with the length of a minimum spanning tree on the same point set; see Figures 2(d)-2(e).

Minimum ink. Let $|\,\mathrm{INK}(P, S)|$ be the length of the shortest curve lying in S connecting all points of P, and let $|\,\mathrm{MST}(P)|$ be the length of the minimum spanning tree of P. The measure is defined as $\frac{|\,\mathrm{MST}(P)|}{|\,\mathrm{INK}(P,S)|}$. Again, 1 indicates the best possible value (though, it does not always correspond to a convex shape); smaller values are worse.

Note that there are advantages and disadvantages of all of the proposed convexity measures, and there are also many other ways to define convexity of shapes or polygons. For example, an "almost" convex shape can be partitioned into a small number of strictly convex polygons. This leads to a measure that asks for a decomposition of a given shape into the minimum number of convex polygons. Such decomposition can be found in polynomial time for a hole-free shape [8], but the problem is computationally hard for arbitrary non-convex shapes [31]. In an attempt to balance theoretical and practical considerations, we focus on the visibility-based and ink-based measures. Similar ink-based criteria are used for constructing LineSets and KelpDiagrams. By minimizing the ink needed for drawing, all of these techniques aim to reduce visual clutter and increase the readability of the representation.

3.2 An Algorithm for Ink Minimization

Here we define and study a new problem motivated by computing contiguous regions with minimum ink. The input consists of n points in the plane, and each point is associated with one of k colors. The Colored Euclidean Steiner Forest (CESF) problem is to connect points of the same color by mutually non-intersecting curves of shortest total length. It is easy to see that in an

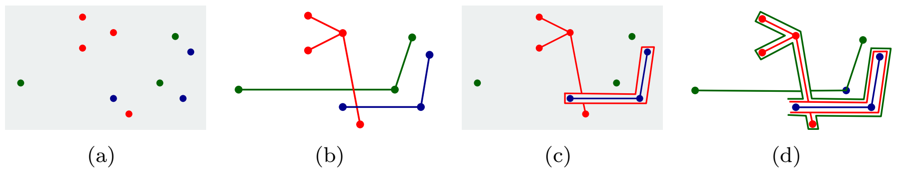


Figure 4: Steps of the algorithm for the CESF problem. (a) An input with $n = 10$ points and $k = 3$. (b) Computing minimum spanning trees. (c) Bounding the tree having the shortest length, and removing red-blue crossings. (d) Merging with the green tree.

optimal solution each curve forms a tree spanning all the points of the corresponding color. In general, the trees may use additional (Steiner) points that do not belong to the original pointset; see Figure 3.

Computing an optimal solution for CESF is NP-hard. This follows directly from the observation that the known NP-complete Minimum Steiner Tree problem is a special case of CESF, in which the input consists of monochromatic points. Next we present a heuristic for CESF and prove that it is an approximation algorithm in the theoretical sense, and hence produces solutions guaranteed to be close to the optimum.

We refer to the minimum spanning tree and the Steiner tree of a set of points P as $\mathrm{MST}(P)$ and $\mathrm{SMT}(P)$, respectively; their lengths are denoted by $|\mathrm{MST}(P)|$ and $|\mathrm{SMT}(P)|$. We use the Steiner ratio, denoted by ρ, which is the supremum of the ratio of the length of a minimum spanning tree to the length of a minimum Steiner tree. Gilbert and Pollak conjectured that $\rho = \frac{2}{\sqrt{3}} \approx 1.15$ [20]. While the conjecture is still open [25], Chung and Graham [10] have shown that ρ is at most ≈ 1.21.

We begin with the description of our algorithm in the bicolored setting, that is, when the input consists of blue and red points. First, we compute a minimum spanning tree of the blue points (ignoring the red ones), and a minimum spanning tree of the red points; see Figure 4(b). If the trees do not intersect, then they form a solution for CESF. Otherwise, we create a red "shell" bounding the blue tree; see Figure 4(c). Note that now all red-blue crossings appear inside the constructed shell. To eliminate the crossings, we remove all portions of the red tree inside the shell; the operation clearly keeps the red tree connected. Finally, the red curve, consisting of the original spanning tree and the constructed shell, can be made into a tree by disconnecting its cycles; see Figure 4(d).

Our general algorithm, for more than 2 colors, works as follows. First, create a minimum tree $\mathrm{MST}(C_i)$ spanning the set of points C_i for $1 \le i \le k$, ignoring points of the other colors. Sort the colors with respect to the length of the corresponding spanning trees. Without loss of generality, we may assume that the resulting order is $C_1, \ldots, C_k$ and $|\mathrm{MST}(C_1)| \le \cdots \le |\mathrm{MST}(C_k)|$. Then the resulting curve for C_1 is the tree $\mathrm{MST}(C_1)$. A curve for each successive color C_i is constructed by adding a “shell” bounding the curve corresponding to C_{i-1}. The length of the shell is exactly $2\sum_{j<i} |\operatorname{MST}(C_j)|$, since it bounds all the spanning trees corresponding to already processed colors; see Figure 4. The length of a curve for C_i is then $|\operatorname{MST}(C_i)| + 2\sum_{j<i} |\operatorname{MST}(C_j)|$.

In order to analyze the algorithm, we denote the amount of ink in the optimal solution by OPT, and the total length of the constructed solution by ALG. An optimal solution induces a curve connecting all points of the same cluster, that is, the solution is a Steiner tree for the set of points (but not necessarily the minimum one). Hence, $\text{OPT} \geq \sum_i |\operatorname{SMT}(C_i)| \geq \sum_i |\operatorname{MST}(C_i)|/\rho$. On the other hand,

$$\text{ALG} \leq \sum_{i=1}^{k} \Big(|\operatorname{MST}(C_i)| + 2\sum_{j=1}^{i-1} |\operatorname{MST}(C_j)| \Big) = \sum_{i=1}^{k} (2k-2i+1) |\operatorname{MST}(C_i)|.$$

For the approximation factor, we get

$$\begin{aligned}
\frac{\text{ALG}}{\text{OPT}} &\leq \frac{\sum_{i=1}^{k}(2k-2i+1)|\operatorname{MST}(C_i)|}{\sum_{i=1}^{k}|\operatorname{MST}(C_i)|/\rho} \\
&= \frac{\sum_{i=1}^{k} k|\operatorname{MST}(C_i)| + \sum_{i=1}^{k}(k-2i+1)|\operatorname{MST}(C_i)|}{\sum_{i=1}^{k}|\operatorname{MST}(C_i)|}\rho \\
&= k\rho + \frac{\sum_{i=1}^{\lfloor k/2\rfloor}(k-2i+1)|\operatorname{MST}(C_i)| + \sum_{i=\lfloor k/2\rfloor+1}^{k}(k-2i+1)|\operatorname{MST}(C_i)|}{\sum_{i=1}^{k}|\operatorname{MST}(C_i)|}\rho \\
&= k\rho + \frac{\sum_{i=1}^{\lfloor k/2\rfloor}(k-2i+1)|\operatorname{MST}(C_i)| - \sum_{i=1}^{\lfloor k/2\rfloor}(k-2i+1)|\operatorname{MST}(C_{k-i+1})|}{\sum_{i=1}^{k}|\operatorname{MST}(C_i)|}\rho \\
&= k\rho + \frac{\sum_{i=1}^{\lfloor k/2\rfloor}(k-2i+1)(|\operatorname{MST}(C_i)| - |\operatorname{MST}(C_{k-i+1})|)}{\sum_{i=1}^{k}|\operatorname{MST}(C_i)|}\rho \leq k\rho.
\end{aligned}$$

Hence, we have the following theorem.

Theorem 1 *There exists a polynomial-time* $(k\rho)$*-approximation for* CESF *for every* $k \geq 1$*.*

4 MapSets

Here we describe MapSets, starting with a high-level overview; see Figure 5. We assume that the input is a set of rectangular shapes (bounding boxes of labels) embedded in the plane along with a clustering. In the first step, we compute spanning mutually non-crossing trees interconnecting centers of rectangles corresponding to the same cluster, while minimizing the total ink needed to draw the trees. In the second step, we modify the trees by adding buffers of free space around the segments of the trees, using a force-directed heuristic. In the third step, we try to optimize the convexity of the resulting regions

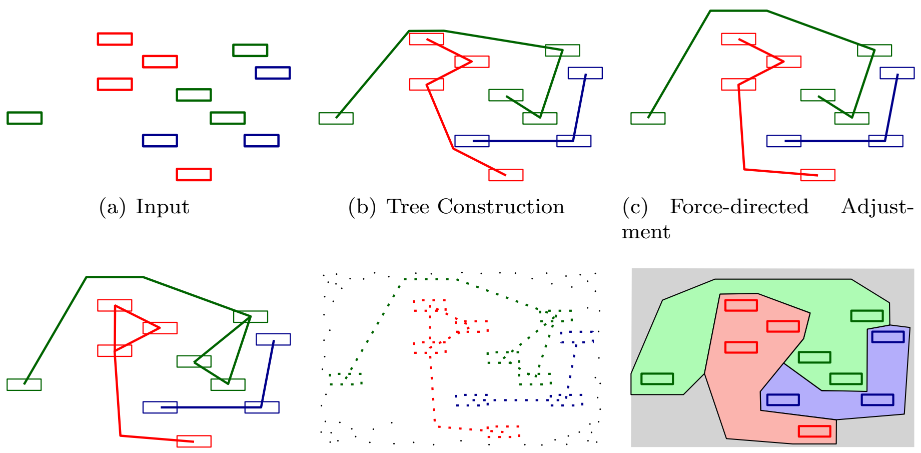

(a) Input (b) Tree Construction (c) Force-directed Adjustment
(d) Edge Augmentation (e) Adding Auxiliary Points (f) Computing Map Regions

Figure 5: Algorithmic pipeline of MapSets.

based on the vertex visibility measure, by adding edges between vertices in the same cluster, while ensuring that edges of different clusters do not cross. In the fourth step, we use the modified trees and the added edges to build contiguous non-overlapping boundaries for all clusters.

Tree construction. In order to construct the trees, we employ the approximation algorithm described in Section 3.2. For each cluster, we first compute a minimum tree spanning the set of rectangle centers, ignoring other clusters. The clusters are then sorted in non-decreasing order by the length of the computed trees and processed in this order. At each step we consider all the pre-computed trees as obstacles that should be avoided when constructing the current tree. The rectangles with different color from the current tree are also treated as obstacles. We compute a visibility graph on the set of obstacles, where the vertices are all the centers and corners of the rectangles, and there is an edge between two vertices if one can draw a straight-line segment without crossing the obstacles. The full visibility graph may have quadratic number of edges and require quadratic time for construction, which can be too slow, especially for interactive applications. Therefore, we utilize a sparse visibility spanner, that is, a subgraph of the visibility graph, approximately preserving shortest paths. We utilize the so called Yao graphs [45], whose construction is based on partitioning the plane around each vertex into a constant number of cones, which is 12 in our implementation. The graph contains at most one edge per cone, and therefore, it has a linear number of edges; such graphs can be constructed efficiently [14]. We then compute shortest paths (of the visibility graph) between every pair of rectangles in the current cluster. From these shortest paths, we compute a minimum spanning tree for the current cluster. We add the tree to the set of obstacles and proceed with the next cluster.

Force-directed adjustment. This step improves the constructed trees. Our goal is to provide some free space around the edges of the trees so as to avoid (1) narrow channels between parts of the same region and (2) region borders lying too close to the input vertex labels. To accomplish this, we consider an adjustment graph G^{adj} in which vertices are the end-points and bends of the constructed trees and edges are maximal straight-line segments of the trees. We then build a force system moving the vertices of G^{adj} that correspond to the bends of the tree. The system relies on the following forces.

- **Vertex-vertex attraction.** We would like to keep the ink of the drawing low. Therefore, for every vertex of G^{adj}, there is a force pushing the vertex towards its neighbor vertices in G^{adj}.
- **Edge-edge repulsion.** This repulsive force attempts to push the edges of G^{adj} apart to provide enough space to draw the regions. In order to compute the force, it is convenient to replace edges of G^{adj} with rectangles of a specified thickness. Then, if two rectangles corresponding to different trees intersect, the force repels them away from each other. This force also ensures that the trees do not overlap and do not intersect during the adjustment process.
- **Edge-label repulsion.** This force prevents edges from being routed too close to the input text labels. Again, it is convenient to consider the edges of G^{adj} as rectangles. The force is applied only if a rectangle occludes a label. In this case, we introduce a repulsive force moving the corresponding vertices of G^{adj} away from the label.

We use iterative refinement similar to that used in drawing graphs with edge bundles [36] to adjust the positions of the vertices of G^{adj} under these three forces: repulsive forces have equal priorities, and the attractive force is weaker. In our experiments, the force system provides the desired buffer of free space around the trees and converges quickly; see Figure 12.

Edge augmentation. In this step we try to optimize the convexity of the regions using the vertex visibility metric. Consider all possible straight-line segments connecting centers of rectangles belonging to the same cluster. Our goal is to select and add as many of these segments as possible, subject to the condition that they do not cross each other. To this end, we construct a graph H in which vertices are the straight-line segments. A segment is added to H only if it does not intersect the trees found in the previous step. Two vertices of H are connected by an edge if the corresponding straight-line segments belong to different clusters and cross each other. Notice that now the problem reduces to the problem of finding a maximum non-crossing (independent) set of segments in the plane. The problem can be solved optimally in polynomial time for two clusters, that is, if $k = 2$. Indeed, in this case the graph H is bipartite, and the size of a maximum independent set in a bipartite graph equals the number of edges in a minimum edge covering by König's theorem. The latter can be found using a maximum matching algorithm. Unfortunately, the general variant is NP-hard even for $k = 3$ [30]. Therefore, for larger values of k, we use a greedy heuristic to solve the problem. At every step, choose a vertex of minimum degree in H, add the vertex to the solution, and remove its neighbors from H. It is well-known that this strategy guarantees an approximation ratio of $(\Delta + 2)/3$ on graphs with maximum degree Δ [22].

Adding auxiliary points and computing map regions. Given the initial placement of the labels and curves connecting the labels from the previous steps, we need explicit regions grouping together labels and curves in the same cluster. To this end, we follow an approach suggested for GMap, which utilizes a Voronoi diagram of the labels' corners [23]. As in GMap, we generate realistic boundaries by adding auxiliary points to the current embedding. There are three types of auxiliary points: (a) random points, sufficiently far away from the set of the input labels, lead to more rounded and thus more realistic region boundaries; (b) random points along bounding boxes of the labels help ensure that the labels are drawn inside the regions; (c) auxiliary points along all the edges constructed on the previous step, that keep the regions connected. The distance between consecutive points on an edge is chosen to be less than the distance to any other point of a different color. After adding the auxiliary points, we compute the Voronoi diagram of the set of all points and merge the Voronoi cells that belong to the points of the same color.

MapSets meets the goals of creating contiguous non-overlapping regions for a given embedded and clustered graphs. The heuristic improvements help make the regions more convex. In Section 5 we compare the proposed method against several existing methods and quantitatively evaluate several aspects of MapSets itself (e.g., ink-minimization, running time of various steps).

Time Complexity. Now we discuss the complexity of our algorithm on an input with n points and k clusters, assuming we can compute distances and intersections between geometric primitives (points, line-segments, rectangles) in constant time. The sparse visibility graph can be constructed in $O(n \log n)$ time and it contains $O(n)$ edges [14]. Therefore, computing all pairwise distances takes $O(n^2 \log n)$ time with Johnson's algorithm. Finding a minimum spanning tree for one cluster takes $O(n^2)$ time using Prim's algorithm. Summing over all clusters, we get $O(kn^2 + n^2 \log n)$ for the first step of our algorithm.

In the iterative force-directed heuristic we compute forces between pairs of vertices, pairs of edges, and between label corners and edges in graph G^{adj}. By construction, the graph contains $O(n)$ vertices and edges; thus, computing all forces takes $O(n^2)$ time for an iteration. Overall, the time complexity of the force-directed heuristic is $O(cn^2)$, where c is the maximum number of iterations in the adjustment ($c = 10$ in our implementation).

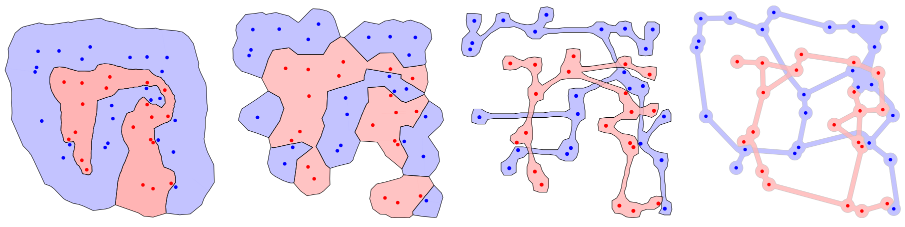


Figure 6: The senator voting graph (the part of the U.S. west of Mississippi). The vertices are senators (red republicans and blue democrats) positioned according to their home-cities.

The complexity of the edge augmentation step is $O(n^3)$, as we may add quadratic number of edges in the greedy process. Adding auxiliary points and creating final regions involves computing a Voronoi diagram of the set of all points. This can be done in $O(n \log n)$ time [23].

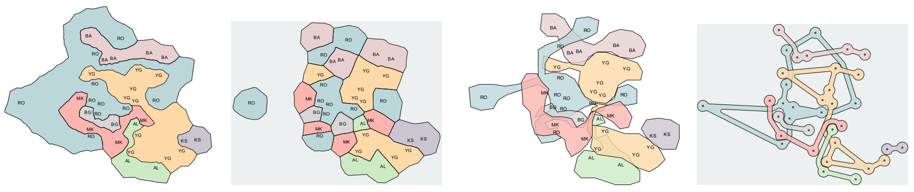
Therefore, the overall time complexity is $O(n^2(k + \log n) + n^3) = O(n^3)$. Actual running times for our example graphs are given in the next section.

5 Experiments

We compare our new algorithm with the existing approaches for map-like visualizations: GMap [23], BubbleSets [11], LineSets [3], and KelpFusion [32]. An implementation of MapSets, GMap, BubbleSets, and LineSets, together with a dataset, is available in an online system at `http://gmap.cs.arizona.edu`.

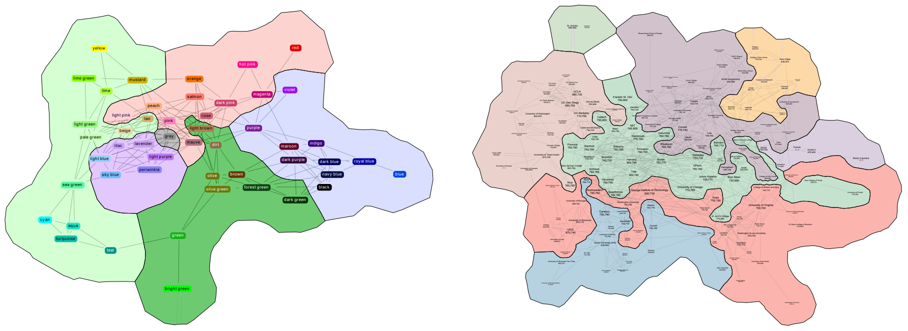
Our first example is the senator voting graph; see Figure 6. The vertices in the graph are the U.S. senators in 2010 positioned according to their home-cities in the U.S. The clustering is based on the political party they represent, red for republicans and blue for democrats. Clearly, both clustering and geographic information of the vertices are fixed and cannot be changed. GMap produces fragmented clusters, while BubbleSets and KelpFusion compute overlapping regions. The result of MapSets is contiguous and non-overlapping, which we believe makes it easier to analyze the distribution of senators over the map.

The second example shows population data in Europe [34]. The original points correspond to genetic data from 1,387 Europeans (but we sampled only 50 vertices corresponding to Eastern Europe for illustration purposes). The positions of the vertices come from the original principal component analysis, based on DNA similarity. As the authors point out, the PCA plot (appropriately rotated) closely matches the geographic outlines of Europe; hence, it is undesirable to change the node positions. The clusters are extracted independently and correspond to the countries of origin of the individuals. Again, only MapSets constructs non-fragmented disjoint regions; see Figure 7. However, it is not clear which visualization technique would be most useful for analysis.



(a) MapSets (b) GMap (c) BubbleSets (d) KelpFusion

Figure 7: The graph of genetic similarities between 50 individuals in Europe. The layout is computed using the principal component analysis, while the clusters correspond to the countries of origin of the individuals.



(a) Colors (b) Universities

Figure 8: Examples of MapSets for two real-world networks. The layout is created using multidimensional scaling, clustering is done using the modularity optimization algorithm. The images are zoomable and have high resolution. (a) The graph is constructed using the 50 most common monitor colors. The edge-weights are defined by the distance in the RGB space between corresponding pairs. (b) The network of the U.S. universities and their average SAT scores. The vertices are universities and edges are constructed based on similarities in admissions.

Further examples of the results of MapSets are given in Figure 8.

Quantitative experiments. We next analyze the performance of our ink minimization algorithm. To this end, we utilize a collection of 10 real-world networks, that are embedded and clustered using the GMap tool with the default setting [23]. Table 1 gives details about the graphs and measurements of our ink saving algorithm. Here, ALG shows the ratio of the total ink of the computed trees to the total length of the minimum spanning trees computed individually for every cluster. In other words, this is an upper bound for the approximation factor achieved by our algorithm on the test cases. Although we can only guarantee approximation factor $k\rho$, in practice the algorithm performs very well, producing solutions that are at most 1.6ρ times worse than optimal. Informally,

graph	Ref.	$\lvert P\rvert$	k	ALG	ALG$_{fd}$
Colors	[28]	50	6	1.002	1.012
GD	[28]	506	23	1.582	1.612
Recipes	[2]	381	15	1.356	1.502
Trade	[23]	211	8	1.101	1.259
Universities	[23]	161	8	1.366	1.443
SODA	[19]	316	11	1.204	1.296
IPL	[19]	336	11	1.337	1.414
SOCG	[19]	500	11	1.492	1.601
TARJAN	[19]	252	16	1.150	1.197
ALGO	[19]	500	5	1.547	1.650

Table 1: Measurements of MapSets on test cases: ALG and ALG$_{fd}$ stand for the ratio between the total ink of the drawing and the total length of the minimum spanning trees after the steps *Tree Construction* and *Force-directed Adjustment*, respectively.

our experiments indicate that the ink minimization often leads to aesthetically more pleasant map visualizations; see Figure 9 for a comparison.

In order to further investigate the approximation factor of our algorithm, we generated a collection of point sets, varying the number of clusters and their spatial distribution. We use $n = 150$ points and $k \in \{2, 3, 5, 10, 15\}$ equally-sized clusters for the dataset. The goal is to create instances of CESF using a parameter, which controls whether the points of the same cluster are spatial neighbors or uniformly distributed in the plane. To this end, we first choose cluster bases by picking k points at random from a unit square. Then for every cluster, n/k points are chosen at random from a disk with radius r centered at the corresponding cluster base. In the experiments, we use $r \in \{0.25, 0.5, 1, 2\}$; here small values of r correspond to spatially disconnected clusters, while larger values correspond to overlapping clusters. We create 10 instances for every combination of k and r, and report the average approximation factor ALG for the test cases; see Figure 10(a) for the results and Figure 10(b) for an example of a run. We observe that the algorithm yields a solution, which is close to the trivial lower bound, if the input clusters are spatially separated from each other. For many real-world examples (e.g., PCA dataset in Figure 7) as well as for graph visualizations in which clustering and embedding are related (e.g., GMap [23]), this is a natural assumption. However, if the input points of different clusters are distributed rather uniformly, then the resulting non-crossing trees may have long detours and be visually unappealing. An interesting observation from Figure 10(a) is that the approximation factor for instances with $k = 2$ is below 1.2, even if the points are chosen randomly from a unit square. This may indicate that our bound for the approximation factor is not tight.

For the force-directed adjustment, we measure ALG$_{fd}$, which is the utilized ink after the step. As expected, the ink increases after the adjustment, but the increase is not significant for both real-world and generates inputs. On

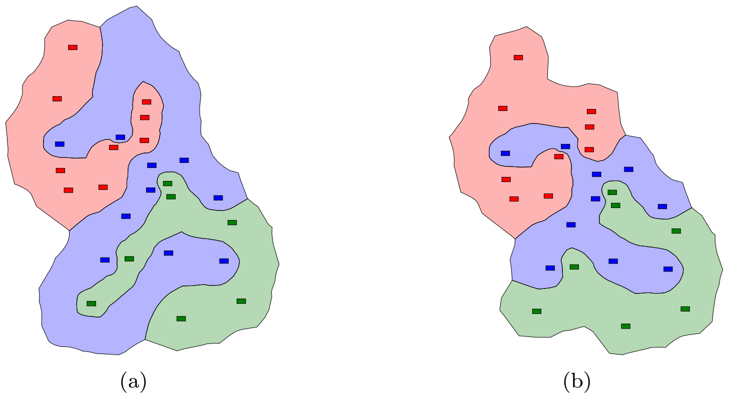
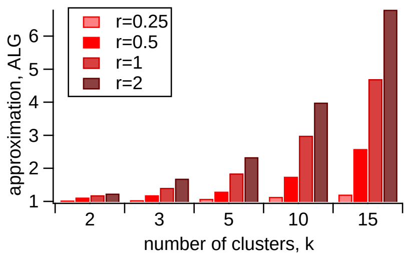


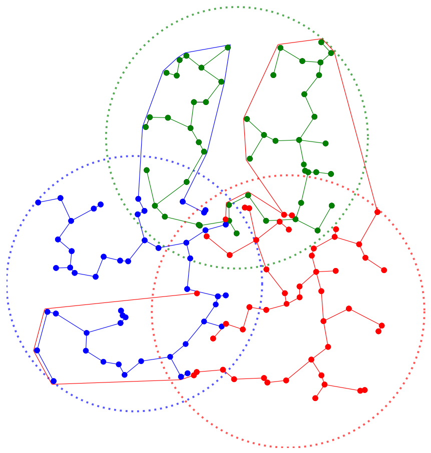
Figure 9: The effect of ink minimization in MapSets. The results computed for the trees with (a) ALG = 1.17 and (b) ALG = 1.06, where ALG is the ratio of the total ink to the total length of the minimum spanning trees.

the other hand, the adjustments improve the quality of the resulting regions. Figure 11 provides an example of the maps computed with and without the heuristic; notice the narrow channels in the figure computed without the adjustments. Finally, we observe that the goal of the step—providing free space around the vertices—can also be achieved by several other techniques. For example, drawing graphs with fat edges [13] and computing non-crossing paths with pre-specified thickness [35]. We leave the study of the usability of such techniques for our setting as an open problem.

Running time. The MapSets algorithm is implemented in C++, whereas for GMap, BubbleSets, and LineSets, we utilize available implementations. We use a machine with Intel i5 3.2GHz and 8GB RAM for measuring running time; see Figure 12. The last two steps, *Adding Auxiliary Points* and *Computing Regions*, are very efficient, taking only a few milliseconds even for the largest graphs, and hence are not included in the chart. The first step, *Tree Construction*, is usually the most time consuming; it is more efficient for nearly contiguous clusters (e.g, Colors) and less efficient for graphs with many fragments (e.g., GD). Although *Edge Augmentation* theoretically has cubic time complexity, it is among the fastest steps in practice, because there are usually not many edges added. Overall, our algorithm processes all the graphs (most with hundreds of vertices) in less than a minute. This is slower than the default GMap algorithm and the LineSets method, but comparable to BubbleSets; see Table 2. Since our algorithm extensively utilizes many primitive geometric operations (e.g., testing for segment intersections), using a specialized geometric library will likely improve the performance.



(a)



(b)

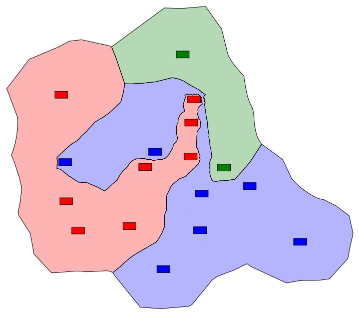
Figure 10: (a) Approximation factor of our algorithm for CESF on the generated dataset. (b) A generated instance with $n = 150$ points, $k = 3$ colors, and $r = 0.5$. The corresponding disks of radius r are shown dashed. For the instance, ALG ≈ 1.38, which indicates that the optimum is at most $1.38\rho < 1.67$ times smaller.

Discussion. Although the results of our initial evaluations seem promising, the comparison of different visualization techniques is not supported by an objective qualitative validation. Arguably, the results of MapSets are easier to analyze when the regions have simple shapes, as in Figure 8(a). We found that in most real-world examples MapSets can serve as a post-processing step for the GMap algorithm, whose resulting regions are often fragmented but still "almost" convex. On the other hand, some input graphs require complex "octopus"-like regions, even in an optimal solution; see Figure 8(b). In these cases it would be better to relax the constraints and to allow either fragmented regions (as in GMap), or intersections between regions (as in BubbleSets and KelpFusion). We leave the question of finding a proper trade-off between the approaches, together with an objective user experiment, for future work.

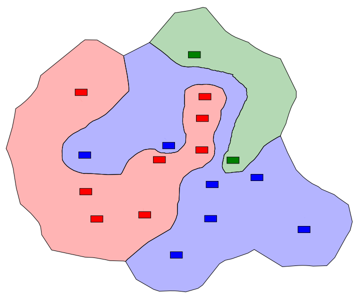
6 Conclusions and Future Work

We designed and implemented a new approach for visualizing embedded and clustered graphs. Unlike existing techniques, our MapSets method always produces contiguous and non-overlapping regions. There are several directions for future work.

We presented a simple $(k\rho)$-approximation algorithm for the CESF problem of ink minimization, where k is the number of clusters and ρ is the Steiner ratio. A natural future direction is to improve the approximation factor. An interesting variant of the problem is when a solution may not contain Steiner points that



(a)



(b)

Figure 11: The effect of the *Force-directed Adjustment* step in MapSets: (a) without the adjustments and (b) with the adjustments. The very thin connection between two blue components is almost invisible without the force-directed adjustments.

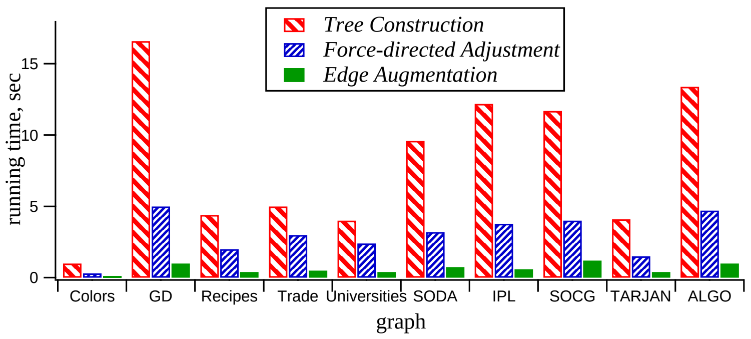


Figure 12: Running times of the different steps of MapSets on the networks described in Table 1. The last two steps (*Adding Auxiliary Points* and *Computing Regions*) take few milliseconds for the graphs and are not included.

are not part of the input. Note that the variant with $k = 1$ corresponds to finding a minimum spanning tree. Is the variant NP-hard for $k > 1$?

The input for our method is a colored point set corresponding to the locations of vertices of a clustered graph. However, in the tree computation step of MapSets, the edges of the graph are not explicitly used. It would be interesting to extend the technique so that it takes the edges of the input graph into account, for example, by requiring regions that span connected subgraphs. An alternative future direction is to allow modification of the given embedding in order to achieve more convex regions.

An in-depth human subjects evaluation is needed to compare map-based visualizations constructed with different approaches considered in the paper. Such a study should cover the spectrum of possible tasks (e.g., node-based tasks, network-based tasks, group-based tasks), consider a range of real-world input graphs (e.g., small/large, sparse/dense, with varying the number of clusters),

graph	MapSets	GMap	BubbleSets	LineSets
Colors	1.3	0.2	1.5	0.3
GD	23.2	0.6	6.4	0.4
Recipes	7.9	0.7	5.7	0.3
Trade	10.2	0.5	4.7	0.3
Universities	8.9	0.3	3.8	0.3
SODA	13.8	0.5	5.1	0.3
IPL	16.9	0.5	7.3	0.3
SOCG	17.6	0.8	11.7	0.6
TARJAN	7.3	0.4	4.2	0.3
ALGO	19.4	0.9	9.4	0.7

Table 2: CPU time (in seconds) taken by different algorithms.

and ensure fair comparison (e.g., font types and sizes, colors).

It would be also worthwhile to carefully evaluate different convexity measures and select one that offers the best balance between ease of computation and visual quality of the resulting regions.

Finally, as mentioned earlier, some of the inputs require complex regions, even in an optimal solution. Hence, a challenging problem is to find a trade-off between contiguity of the resulting regions and their visual complexity. How difficult is to minimize the number of "almost" convex fragments for a given pointset?

Acknowledgements

This work is supported in part by NSF grants CCF-1115971 and DEB 1053573. We thank the authors of [34] for the DNA dataset. The drawings of KelpFusion are courtesy of the authors of [32].

References

[1] P. K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete Comput. Geom.*, 6(1):407–422, 1991. doi:10.1007/BF02574698.

[2] Y.-Y. Ahn, S. E. Ahnert, J. P. Bagrow, and A.-L. Barabási. Flavor network and the principles of food pairing. *Scientific reports*, 1, 2011. doi:10.1038/srep00196.

[3] B. Alper, N. H. Riche, G. Ramos, and M. Czerwinski. Design study of LineSets, a novel set visualization technique. *IEEE Trans. Visual. Comput. Graphics*, 17(12):2259–2267, 2011. doi:10.1109/TVCG.2011.186.

[4] S. Arora and K. Chang. Approximation schemes for degree-restricted MST and red–blue separation problems. *Algorithmica*, 40(3):189–210, 2004. doi:10.1007/s00453-004-1103-4.

[5] M. J. Atallah and D. Z. Chen. On connecting red and blue rectilinear polygonal obstacles with nonintersecting monotone rectilinear paths. *Int. J. Comput. Geom. Appl.*, 11(04):373–400, 2001. doi:10.1142/S0218195901000547.

[6] K. W. Boyack, R. Klavans, and K. Börner. Mapping the backbone of science. *Scientometrics*, 64:351–374, 2005. doi:10.1007/s11192-005-0255-6.

[7] H. Byelas and A. Telea. Towards realism in drawing areas of interest on architecture diagrams. *Journal of Visual Languages & Computing*, 20(2):110–128, 2009. doi:10.1016/j.jvlc.2008.09.001.

[8] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. In *Symposium on Theory of Computing*, pages 38–48. ACM, 1979. doi:10.1145/800135.804396.

[9] S. C. Chow. *Generating and drawing area-proportional Euler and Venn diagrams*. PhD thesis, University of Victoria, 2007.

[10] F. Chung and R. Graham. A new bound for Euclidean Steiner minimal trees. *Annals of the New York Academy of Sciences*, 440(1):328–346, 1985. doi:10.1111/j.1749-6632.1985.tb14564.x.

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

[12] K. Dinkla, M. J. van Kreveld, B. Speckmann, and M. A. Westenberg. Kelp diagrams: Point set membership visualization. *Comput. Graph. Forum*, 31(3pt1):875–884, 2012. doi:10.1111/j.1467-8659.2012.03080.x.

[13] C. A. Duncan, A. Efrat, S. Kobourov, and C. Wenk. Drawing with fat edges. *Int. J. Found. Comput. S.*, 17(05):1143–1163, 2006. doi:10.1142/S0129054106004315.

[14] T. Dwyer and L. Nachmanson. Fast edge-routing for large graphs. In D. Eppstein and E. Gansner, editors, *Graph Drawing*, volume 5849 of *Lecture Notes Comput. Sci.*, pages 147–158. Springer, 2010. doi:10.1007/978-3-642-11805-0_15.

[15] A. W. F. Edwards. *Cogwheels of the mind: the story of Venn diagrams.* JHU Press, 2004.

[16] A. Efrat, Y. Hu, S. Kobourov, and S. Pupyrev. MapSets: Visualizing embedded and clustered graphs. In C. Duncan and A. Symvonis, editors, *Graph Drawing*, volume 8871 of *Lecture Notes Comput. Sci.*, pages 452–463. Springer, 2014. doi:10.1007/978-3-662-45803-7_38.

[17] S. Fabrikant, D. Monteilo, and D. M. Mark. The distance-similarity metaphor in region-display spatializations. *IEEE Comput. Graph. Appl.*, 26(4):34–44, 2006. doi:10.1109/MCG.2006.90.

[18] Q.-W. Feng, R. Cohen, and P. Eades. Planarity for clustered graphs. In P. Spirakis, editor, *ESA*, volume 979 of *Lecture Notes Comput. Sci.*, pages 213–226. Springer, 1995. doi:10.1007/3-540-60313-1_145.

[19] D. Fried and S. G. Kobourov. Maps of computer science. *Pacific Visualization Symposium*, pages 113–120, 2014. doi:10.1109/PacificVis.2014.47.

[20] E. Gilbert and H. Pollak. Steiner minimal trees. *SIAM J. Appl. Math.*, 16(1):1–29, 1968. doi:10.1137/0116001.

[21] M. Gronemann and M. Jünger. Drawing clustered graphs as topographic maps. In W. Didimo and M. Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes Comput. Sci.*, pages 426–438. Springer, 2013. doi:10.1007/978-3-642-36763-2_38.

[22] M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997. doi:10.1145/195058.195221.

[23] Y. Hu, E. R. Gansner, and S. G. Kobourov. Visualizing graphs and clusters as maps. *IEEE Comput. Graphics and Appl.*, 30(6):54–66, 2010. doi:10.1109/MCG.2010.101.

[24] F. Hurtado, M. Korman, M. Kreveld, M. Löffler, V. Sacristán, R. Silveira, and B. Speckmann. Colored spanning graphs for set visualization. In S. Wismath and A. Wolff, editors, *Graph Drawing*, volume 8242 of *Lecture Notes Comput. Sci.*, pages 280–291. Springer, 2013. doi:10.1007/978-3-319-03841-4_25.

[25] A. Ivanov and A. Tuzhilin. The Steiner ratio Gilbert-Pollak conjecture is still open. *Algorithmica*, 62(1–2):630–632, 2012. doi:10.1007/s00453-011-9508-3.

[26] R. Jianu, A. Rusu, Y. Hu, and D. Taggart. How to display group information on node-link diagrams: An evaluation. *IEEE Trans. Visual. Comput. Graphics*, 20(11), 2014. doi:10.1109/TVCG.2014.2315995.

[27] G. Kanizsa and W. Gerbino. Convexity and symmetry in figure-ground organization. *Vision and Artifact*, pages 25–32, 1976.

[28] S. G. Kobourov, S. Pupyrev, and P. Simonetto. Visualizing graphs as maps with contiguous regions. In *Eurographics Conference on Visualization*, pages 31–35, 2014. doi:10.2312/eurovisshort.20141153.

[29] T. Kohonen. *Self-organizing maps*, volume 30 of *Springer Series in Information Sciences*. Springer, 2001.

[30] J. Kratochvíl and J. Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Comment. Math. Univ. Carolinae*, 31(1):85–93, 1990.

[31] A. Lingas. The power of non-rectilinear holes. In M. Nielsen and E. Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *Lecture Notes Comput. Sci.*, pages 369–383. Springer, 1982. doi:10.1007/BFb0012784.

[32] W. Meulemans, N. Riche, B. Speckmann, B. Alper, and T. Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE Trans. Visual. Comput. Graphics*, 19(11):1846–1858, 2013. doi:10.1109/TVCG.2013.76.

[33] J. S. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and J. Urrutia, editors, *Handbook of computational geometry*, pages 633–701. Elsevier, 2000. Section 7.1.

[34] Novembre et al. Genes mirror geography within Europe. *Nature*, 456(7218):98–101, 2008. doi:10.1038/nature07331.

[35] V. Polishchuk and J. S. Mitchell. Thick non-crossing paths and minimum-cost flows in polygonal domains. In *Symposium on Computational Geometry*, pages 56–65. ACM, 2007. doi:10.1145/1247069.1247079.

[36] S. Pupyrev, L. Nachmanson, S. Bereg, and A. Holroyd. Edge routing with ordered bundles. In M. van Kreveld and B. Speckmann, editors, *Graph Drawing*, volume 7034 of *Lecture Notes Comput. Sci.*, pages 136–147. Springer, 2012. doi:10.1007/978-3-642-25878-7_14.

[37] D. Purves and R. B. Lotto. *Why we see what we do: An empirical theory of vision.* Sinauer Associates, 2003. doi:10.1162/089892903770007452.

[38] N. H. Riche and T. Dwyer. Untangling Euler diagrams. *IEEE Trans. Visual. Comput. Graphics*, 16(6):1090–1099, 2010. doi:10.1109/TVCG.2010.210.

[39] B. Saket, P. Simonetto, S. Kobourov, and K. Börner. Node, node-link, and node-link-group diagrams: An evaluation. *IEEE Trans. Visual. Comput. Graphics*, 20(12):2231–2240, 2014. doi:10.1109/TVCG.2014.2346422.

[40] P. Simonetto, D. Auber, and D. Archambault. Fully automatic visualisation of overlapping sets. *Comput. Graph. Forum*, 28(3):967–974, 2009. doi:10.1111/j.1467-8659.2009.01452.x.

[41] A. Skupin. A cartographic approach to visualizing conference abstracts. *IEEE Comput. Graph. Appl.*, 22(1):50–58, 2002. doi:10.1109/38.974518.

[42] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Thomson-Engineering, 2007. doi:10.1007/978-1-4899-3216-7.

[43] G. Stapleton, P. Rodgers, J. Howse, and L. Zhang. Inductively generating Euler diagrams. *IEEE Trans. Visual. Comput. Graphics*, 17(1):88–100, 2011. doi:10.1109/TVCG.2010.28.

[44] S.-i. Tokunaga. Intersection number of two connected geometric graphs. *Inform. Process. Lett.*, 59(6):331–333, 1996. doi:10.1016/0020-0190(96)00124-X.

[45] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. doi:10.1137/0211059.

[46] J. Zunic and P. L. Rosin. A convexity measurement for polygons. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26:173–182, 2002. doi:10.1109/TPAMI.2004.19.