DOI: 10.1111/cgf.14322
Eurographics Conference on Visualization (EuroVis) 2021
R. Borgo, G. E. Marai, and T. von Landesberger
(Guest Editors)

Volume 40 (2021), Number 3

ClusterSets: Optimizing Planar Clusters in Categorical Point Data

J. Geiger[1], S. Cornelsen[2], J.-H. Haunert[3], P. Kindermann[1,4], T. Mchedlidze[5], M. Nöllenburg[6], Y. Okamoto[7], and A. Wolff[1]

[1] Universität Würzburg, Würzburg, Germany
[2] University of Konstanz, Konstanz, Germany
[3] University of Bonn, Bonn, Germany
[4] Universität Trier, Trier, Germany
[5] Utrecht University, Utrecht, The Netherlands
[6] TU Wien, Vienna, Austria
[7] University of Electro-Communications, Tokyo, Japan

Figure 1: *Part of a larger set of points representing facilities of the University of Bonn, where colors of points reflect department memberships. Left: Voronoi diagram with Voronoi cells colored with same hues as corresponding points. Right: ClusterSets visualization with edges defining clusters as selected by our greedy heuristic (and by a post-processing step where intra-cluster edges may cross each other), a line Voronoi diagram of those edges, and polygonal representations of the clusters. ClusterSets produces larger contiguous regions (compared to the Voronoi diagram) but still reflects the proximity relationships among the points – see, e.g., the three blue points split up into two clusters.*

Abstract

In geographic data analysis, one is often given point data of different categories (such as facilities of a university categorized by department). Drawing upon recent research on set visualization, we want to visualize category membership by connecting points of the same category with visual links. Existing approaches that follow this path usually insist on connecting all members of a category, which may lead to many crossings and visual clutter. We propose an approach that avoids crossings between connections of different categories completely. Instead of connecting all *data points of the same category, we subdivide categories into smaller, local clusters where needed. We do a case study comparing the legibility of drawings produced by our approach and those by existing approaches.*

In our problem formulation, we are additionally given a graph G on the data points whose edges express some sort of proximity. Our aim is to find a subgraph G' of G with the following properties: (i) edges connect only data points of the same category, (ii) no two edges cross, and (iii) the number of connected components (clusters) is minimized. We then visualize the clusters in G'. For arbitrary graphs, the resulting optimization problem, Cluster Minimization, is NP-hard (even to approximate). Therefore, we introduce two heuristics. We do an extensive benchmark test on real-world data. Comparisons with exact solutions indicate that our heuristics do astonishing well for certain relative-neighborhood graphs.

1. Introduction

Recent research on set visualization has focused on computing visual set representations based on visual links that connect elements within a set. Often, the sets correspond to categories of data entities. When visualizing sets of non-spatial entities, e.g., sets of composers of different music genres, every entity can be represented as a point symbol that can be freely placed in the visualization. In contrast, in this paper we present a new method that we call *ClusterSets* to visualize sets of points in spatial data, such as facilities of a university categorized by department (Fig. 1) or points of interest of different categories, e.g., hotels or restaurants (Fig. 9).

Similar to the Bubble Sets method by Collins et al. [CPC09], our ClusterSets method assumes that the points have fixed positions. This is highly relevant in *geovisualization*, where the positions of point symbols in a map correspond to geographical locations. Moreover, one may use Bubble Sets or ClusterSets to augment a primary visualization whose layout has been chosen for a good reason, thus respecting the *spatial rights* [CPC09] of the primary visualization. We take the idea of spatial rights one step further, by granting the visualization a *right for locality*. This means that not necessarily all points of the same category need to be connected but that the proximity relationships among the points should be taken into consideration as well when deciding which points to connect. Two spatial clusters of the same category, for example, should not be visually linked if they are clearly separated by a third cluster of a different category. This is most important if the primary visualization is a map, which usually aims to reveal spatial patterns and relations [Kra09]. Violating the right for locality by adding strong visual links between points that do *not* belong to the same same spatial cluster could reduce the usability of the map. Hence, we developed ClusterSets to mediate between (i) the visualization of sets defined with categories of points and (ii) the computation and visualization of clusters based on spatial proximity.

Relaxing the requirement of connecting all points of the same category allows us not only to maintain the right for locality but also to keep the visual clutter low. In particular, by computing spatially disjoint clusters we avoid overlapping visual representations of sets, which are prevalent in set visualizations based on Bubble Sets or related methods. In fact, our approach avoids crossings between connections of different categories completely. Instead of connecting all data points of the same category, it subdivides categories into smaller, local and planar clusters where needed.

Our clustering algorithms construct spanning subgraphs in suitably chosen proximity graphs, whose edges define which pairs of points are sufficiently close local neighbors that can be used for growing clusters. Since these proximity graphs are in general not plane, but overlapping clusters in the solution would negatively impact their readability, we require solutions to be plane, leading to the following optimization problem for geometric graphs, that is, for graphs whose vertices are points in the plane and whose edges are straight-line segments connecting the vertices.

Definition 1 (CLUSTER MINIMIZATION) Let G be a geometric graph that may contain edge crossings. Find a plane subgraph of G that has as few connected components (clusters) as possible.

Note that this definition does not take different vertex categories into account. Instead, we can simply delete all edges between vertices of different categories; see Fig. 3b. Then our clustering problem for categorical point data becomes exactly CLUSTER MINIMIZATION. Consequently, in the context of this problem, a crossing between two edges is forbidden, no matter whether the two edges belong to different categories or the same category. Since adding intra-cluster edges may help to improve the visual representations of clusters, however, we will introduce a post-processing step to augment solutions of CLUSTER MINIMIZATION with edges that may cross other edges of the same cluster.

Results Our contribution is two-fold. First, we deal with the challenging combinatorial problem CLUSTER MINIMIZATION. Second, we present ways to visualize any clustering of the input points if (i) the clustering is a refinement of the given point categories and (ii) no edge of one cluster crosses any edge of another cluster.

Concerning our first contribution, it turns out that, CLUSTER MINIMIZATION is NP-hard, even if each connected component of the input graph contains exactly two vertices (maximum independent set of segment intersection graphs [KN90]). Our problem is even hard to approximate (Section 8). Therefore, we introduce two greedy heuristics (Section 5.1). For these, we give examples of graph families on which they perform arbitrarily badly. Still the heuristics perform surprisingly well on real-world data (Section 7). For the case of *1-plane graphs*, i.e. graphs in which each edge is crossed by at most one other edge, the heuristics actually yield the optimum (Section 5.1). This graph family includes the graph of all useful order-1 Delaunay edges [GHvK02]. To evaluate the two heuristics, we devised an exact solution for CLUSTER MINIMIZATION based on integer linear programming (Section 5.2).

Our second contribution is a case study (Section 6) where we explore ways to visualize clusterings with disjoint clusters. Consider first the following naive baseline method. Compute the Voronoi diagram [dBCvKO08] of the input points and color each cell according to the category of the corresponding point site; see Fig. 1 (left). This is the same as using the Delaunay triangulation as proximity graph and removing all edges between points of different categories. This would yield a large number of clusters. Therefore, we use a denser proximity graph (see Section 3.2), apply one of our clustering algorithms, and then use any of the following three visualization methods.

The *tree method* simply takes the plane edge sets of the clusters and highlights them using thick segments; clusters that belong to the same category use the same color. The other two methods add all intra-cluster edges back in under the condition that they do not cross edges of other clusters. This connects the clusters more densely. The *line-Voronoi method* simply computes the *line Voronoi diagram* [dBCvKO08], where the Voronoi sites are not the vertices but the straight-line edges in the clustering. Finally, the *polygon method* traces the outline of each cluster using thick line segments. Each "areal" cluster is filled with a lighter color that can be transparent as in Fig. 3c. Adding back edges can lead to enclaves; we avoid this by additional checks. For a comparison of the three methods, see Fig. 9 in our case study (Section 6). Clearly, our clusterings can also be used as input for other visualization methods. Moreover, we combine the visualization methods as in Fig. 1 (right), where the line-Voronoi method and polygon method are combined.

2. Related Work

Finding and visualizing clusters in point sets is a common task in visual data analytics to reveal, explore, and communicate their spatial patterns.

Several methods have been proposed in the literature to compute cluster visualizations. Generally speaking, cluster visualization is a special case of set visualization [AMA*16], where the clusters form a partition of the elements with disjoint sets, rather than with sets having more complex intersection patterns. Set visualization itself can be grouped into *abstract set visualization*, where elements do not have initial spatial locations (e.g., movies and their genres), and into *spatial set visualization*, where all elements come with a predefined position (e.g., points of interest in a map or points in a scatter plot). Our paper falls into the latter category, where the most prominent existing methods are overlay techniques that aim to visually group elements of the same cluster into coherent regions, often making use of the Gestalt principles for grouping.

The first group of techniques are region-based overlay techniques, which enclose the points of each cluster by a closed curve and typically fill the regions by distinct colors. Bubble Sets [CPC09] is a well-known region-based technique that draws isocontours surrounding the points of each cluster as a contiguous hull minimizing (but not excluding) overlap with other hulls. GMap [GHK10] computes Voronoi-based cluster shapes for the given point locations with the aim to create resemblance to political maps. However, depending on the properties of the point configurations, it may create fragmented, disconnected cluster shapes from the computed Voronoi cells. MapSets [EHKP15] uses a similar map metaphor as GMap, but was designed with the goal to obtain connected cluster shapes based on a set of plane spanning trees for the different clusters. Both GMap and MapSets properly subdivide the canvas and hence do not create cluster overlaps.

A second group of overlay techniques are line-based overlays, which connect the points of each cluster into a spanning graph. The first such technique is LineSets [AHRRC11], which uses a traveling salesperson heuristic to find a short path for each cluster that visits all its points. The resulting lines are rendered as colored Bézier splines. However, since each path is computed independently, different lines may mutually cross each other in visually complex ways. Kelp diagrams [DvKSW12] generalize the idea of LineSets and compute sparse spanning graphs for possibly overlapping sets indicated as a nested containment. Kelp diagrams typically produce fewer line crossings than LineSets, but are not guaranteed to be planar. In contrast, Castermans et al. [CvGM*19] require planarity of their tree-based spatial set visualization technique, which aims to minimize the total edge length of the plane spanning trees. The underlying computational problem is shown as NP-hard; heuristic and exact ILP-based algorithms are proposed. The planarity requirement comes at the cost that a valid solution does not always exist, which limits the method's applicability.

An interesting compromise between region-based and line-based techniques is KelpFusion [MHRS*13]. This hybrid technique uses local hulls for denser configurations of points in the same cluster and lines to bridge larger distances between different parts of a cluster. With its parameterization it can be customized to create cluster shapes that range from a spanning tree to a convex hull. While all of the line-based and hybrid techniques create connected cluster shapes, they do suffer from various degrees of possibly disturbing crossings and visual clutter. With ClusterSets, we study the other end of the trade-off between connectivity and planarity. Rather than enforcing a single connected cluster per category in the data, we require crossing-free cluster layouts at the cost of creating possibly disconnected clusters of the same categories. However, we minimize the number of disconnected clusters per category.

For the cartographic visualization of categorical point data, Bertin's system of visual variables [Ber11] is usually applied, often by representing different categories with point symbols of different shapes or hues. Recent research has dealt with special symbols, termed Micro Diagrams, that display multiple categories [GB20]. Generally, the (dis-)similarity of point symbols can help humans discover spatial patterns such as a group of multiple points of the same category. It has also been shown, however, that connectivity has a higher influence on perceptual grouping than similarity [BP02]. With our graph-based visualization method we thus contribute to making patterns in categorical point data more explicit.

Finally, we note that our method can be classified as a neighborhood-based clustering algorithm, of which there are too many to provide a comprehensive review. Nevertheless, we refer to basic concepts [Zah71], recent work on the topic including a discussion of the state of the art [AOA19], and the popular DBScan algorithm [EKASX96], which is based on an ε-neighborhood graph. A special characteristic of our method is that it is not restricted to a specific neighborhood graph (although we suggest using β-skeletons) and that the produced clustering is defined by a crossing-free spanning forest (a forest is a cycle-free graph, i.e., a union of trees). This is particularly useful to avoid visual clutter.

3. ClusterSets

ClusterSets is a combined clustering and visualization approach for categorical point data. In this section we present our design goals as well as the underlying modular pipeline.

3.1. Design Goals

The challenge in visualizing categorical point data is that there are multiple diverging optimization goals. On the one hand, one can use symbols of different colors or shapes to indicate the categorical information. Such visualizations show every point as a singleton but especially in dense point sets it can become difficult to detect spatial grouping patterns visually. On the other hand, overlay set visualization methods apply the Gestalt principle of enclosure to highlight all points of the same category as a common geometric shape. But these cluster regions may easily produce clutter if the categories are not well separated in the point set. Hence, we decided to develop a geometric clustering strategy that aims to partition all points of the same category into a minimum number of pairwise disconnected clusters such that no two clusters overlap, regardless of their category. Such a clustering can then be visualized as a collection of planar regions, which avoids the visual clutter of region overlaps but still applies the enclosure principle as much as possible. With such a cluster visualization approach the number of clusters per category adapts to the spatial properties of the distribution

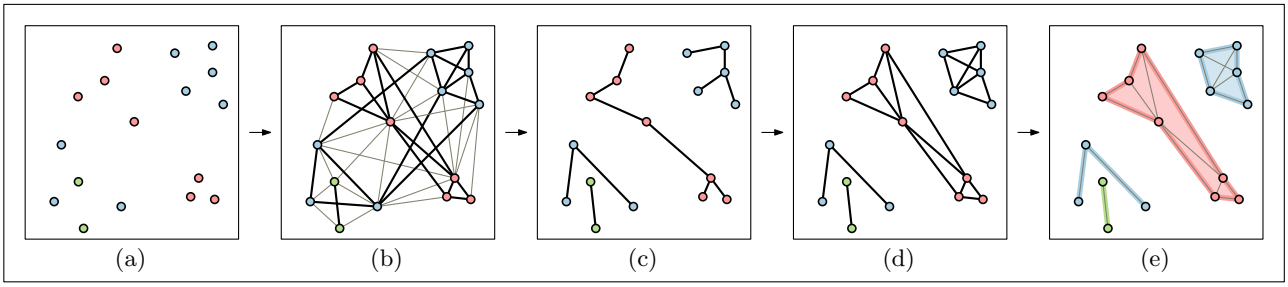


Figure 2: *Four steps of the ClusterSets pipeline. (a) Input point data with three categories (red, blue, green), (b) proximity graph (black edges are intra-category edges), (c) planar cluster spanning forest, (d) edge augmentation, (e) final layout with four planar clusters.*

of the categories in the point data. With this in mind we define the following four design goals:

1. Each category is represented by a distinct color.
2. Each cluster is a subset of points from the same category that can be enclosed/connected into a single geometric component that does not overlap with the component of any other cluster.
3. For each point in a cluster there is a sufficiently close point in the same cluster to which it can be connected by an edge in a suitable proximity graph.
4. The number of clusters per category should be small.

3.2. ClusterSets Pipeline

ClusterSets is implemented as a modular 4-step pipeline that realizes the above design goals. This pipeline is shown in Fig. 2.

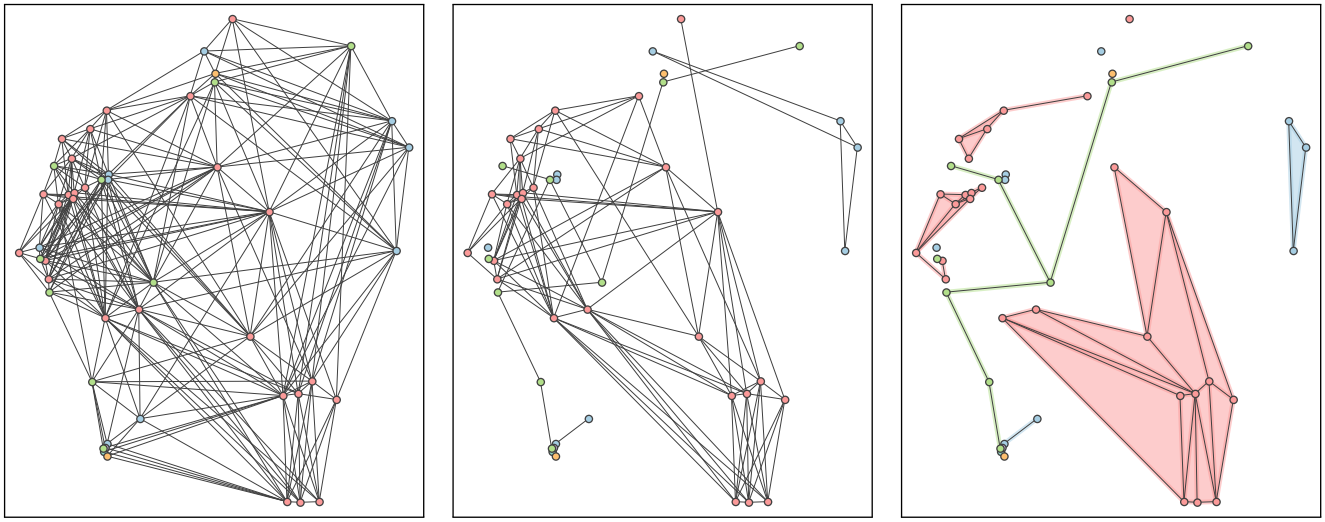
1. **Proximity graph.** In the first step we create a suitable proximity graph for the point set. This graph guides how clusters can be formed in the subsequent steps. The principle of a proximity graph is to define edges between any two points that can be considered sufficiently close to each other with respect to the surrounding points. Examples of such proximity graphs are the Delaunay triangulation, the Gabriel graph [MS80], or the β-skeleton, which we use in our implementation. Since clusters must have a unique category, we remove all edges with endpoints in different categories before we proceed to the next step.
2. **Planar spanning forest.** In the second step we compute a planar spanning forest in the proximity graph that minimizes the number of connected components. Since this problem is NP-hard [JW93, KN90], we propose two simple heuristics and an exact integer linear program in Section 5. The result of this step is a collection of crossing-free subgraphs of the proximity graph. They define a set of homogeneous clusters.
3. **Edge augmentation.** While we can use the spanning trees directly to visualize the clusters, this would result in a purely line-based overlay. Hence we define an optional edge augmentation step, which enriches each cluster by additional edges from the proximity graph such that no two edges from different clusters cross each other. The contours of these spanning graphs yield hull polygons for each cluster.
4. **Rendering.** The last step visualizes the clusters using one of three representations. In particular, we propose a *tree representation* that shows a spanning tree for each cluster, a *line-Voronoi representation* partitioning the plane into regions of different colors, and a *polygon representation* showing the area enclosed by selected edges as colored polygons. We use thicker colored edges for the contours of the cluster regions, and a lighter shade of the same color to fill the polygons if applicable. Singleton clusters receive a larger colored halo to highlight their category.

4. Proximity Graphs

The first step of our pipeline computes a suitable proximity graph, in which we can then construct planar clusters. An example for a possible proximity graph is the Delaunay triangulation [dBCvKO08], which gives reasonably-well-defined clusters – and *no* edge crossings – but the clusters tend to be rather small. Denser graphs contain more connections to choose from in order to obtain larger clusters. Such graphs can be obtained via the set of *useful higher-order Delaunay edges* [GHvK02]. For example, the set of *useful order-1 Delaunay edges* induces a 1-plane graph. For our case study and our experimental analysis, we opted, however, for another type of proximity graphs, the β-*skeletons*. In the β-skeleton for some $0 < \beta \leq 1$, two points p, q are connected if there is no third point in the lune defined as the intersection of the two disks with diameter $|pq|/\beta$ passing through p and q. Depending on the chosen parameter β, the skeletons can range from a complete graph ($\beta = 0$) to the *Gabriel graph* ($\beta = 1$), a subgraph of the Delaunay triangulation that was introduced by Matula and Sokal [MS80] in the context of geographic cluster analysis. The fact that we can directly control the density of these proximity graphs via a single parameter makes them very handy for us. Larger values of β are possible, but yield rather sparse results and are therefore of minor interest for our application. An example of a 0.5-skeleton is given in Fig. 3a. As we will show in our experimental evaluation (Section 7), denser proximity graphs mean fewer (but less compact) clusters; see Fig. 8. Setting $\beta = 0.5$ yields a good trade-off. We note that edges in the proximity graph that connect points from two different categories are not useful for cluster formation. Hence we delete all these inter-category edges before proceeding to the next step (see Fig. 3b).

5. Clustering Algorithms

As we noted in the introduction, CLUSTER MINIMIZATION is NP-hard even in the case that each category consists of only two vertices [KN90]. This complexity result justifies that, in the remainder



(a) β-*skeleton for* $\beta = 0.5$ **(b)** *removal of all bichromatic edges* **(c)** *an optimal solution*

Figure 3: *Input and output for* CLUSTER MINIMIZATION *given a 50-point instance from the OSM data set.*

of this section, we present two fast heuristics for the problem (thus accepting suboptimal solutions) and an exact ILP formulation (thus accepting exponential running time).

5.1. Heuristics

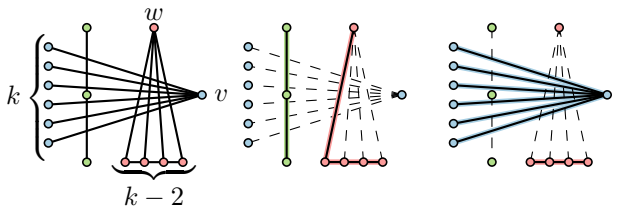
We propose two heuristics for the cluster minimization problem. The GREEDY heuristic keeps joining subclusters by edges that are crossed by as few other edges as possible. The REVERSE GREEDY heuristic keeps removing edges that are crossed by many edges. We show that in theory both heuristics can perform arbitrarily bad and that neither is better than the other (Figs. 4–5). More precisely, given a geometric graph $G = (V, E)$, the two heuristics compute a plane spanning subgraph $G' = (V, E')$ of G. Starting with $E' = \emptyset$, we empty E by repeating the following three steps.

GREEDY: (1) Keep removing edges from E whose end vertices belong to the same connected component in G'. (2) Move an edge e from E to E' that is crossed by the minimum number of edges. (3) Remove all edges from E that used to cross e.

REVERSE GREEDY: (1) Keep removing edges from E whose end vertices belong to the same connected component in G'. (2) Move all edges from E to E' that are not crossed by any other edge of E unless they would close a cycle in G'. (3) Remove an edge e from E that is crossed by the maximum number of edges.

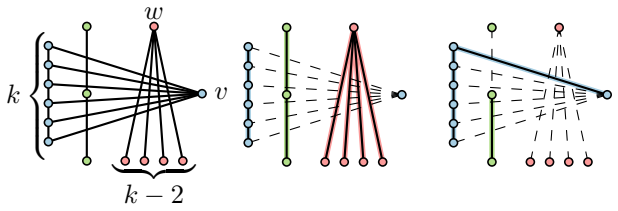
In both cases, the result is a crossing-free spanning forest. Using a sweep-line algorithm for computing the crossings [BO79] and a Union-Find data structure for maintaining the connected components, the run time of GREEDY and REVERSE GREEDY on input graphs with m edges is in $\mathcal{O}(m^2)$.

For the remainder of this section, we focus on *1-plane* graphs,



(a) graph, $k > 4$ even (b) GREEDY (c) REVERSE GREEDY

Figure 4: *Both* GREEDY *and* REVERSE GREEDY *first pick the* $k-3$ *uncrossed (bottommost) edges. Then (b)* GREEDY *picks the two vertical edges that are crossed* $k/2$ *times and removes the edges incident to v. (c)* REVERSE GREEDY *removes all edges incident to w and then the two vertical edges. Finally, uncrossed edges are added resulting in a graph with* $k+3$ *(*GREEDY*) or* 6 *(*REVERSE GREEDY*) connected components.*



(a) graph, $k > 4$ even (b) GREEDY (c) REVERSE GREEDY

Figure 5: *Both,* GREEDY *and* REVERSE GREEDY *first pick the* $k-1$ *uncrossed (leftmost) edges. Then (b)* GREEDY *picks the two vertical edges that are crossed* $k/2$ *times and removes the edges incident to v. (c)* REVERSE GREEDY *removes all edges incident to w and one among the two vertical edges. Then one of the uncrossed edge incident to v is picked and the others removed. Finally, uncrossed edges are added resulting in a graph with* 4 *(*GREEDY*) or* $k+2$ *(*REVERSE GREEDY*) connected components.*

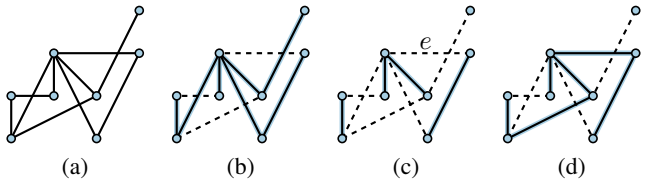


Figure 6: GREEDY *solves* CLUSTER MINIMIZATION *on 1-plane graphs not always optimally: (a) input graph, (b) optimal solution, (c)* GREEDY *first chooses the four uncrossed edges. Next it might pick edge e, which would lead to suboptimal solution (d).*

i.e., graphs where each edge is crossed at most once. Admittedly, 1-plane graphs are a rather special case, but – see the discussion about useful order-1 Delaunay edges in Section 4 – it is not irrelevant.

Observe that minimizing the number of connected components corresponds to maximizing the number of edges when looking for crossing-free spanning forests. Thus, it is possible to use some heavy machinery from algorithm theory and formulate CLUSTER MINIMIZATION for 1-plane graphs as the intersection of two *matroids* [KV18]: the set of all cycle-free edge sets and the set of all crossing-free edge sets. Hence, cluster minimization can be solved optimally in $\mathcal{O}(n^3 \log n)$ time on 1-plane graphs with n vertices [CLS*19].

As it turns out, however, for 1-plane graphs, the two heuristics are identical and yield an optimal result under the following condition. We need to allow the merging of clusters at crossings between edges that join vertices of the same category, i.e., we have to replace initial crossings of edges of the same connected component by 4-stars. (Note that this modification may change the optimum.) Figure 6 shows that the modification is necessary.

Theorem 1 Both GREEDY and REVERSE GREEDY solve the cluster minimization problem optimally in $\mathcal{O}(n \log n)$ time on 1-plane graphs with n vertices in which no two edges of the same connected component cross.

Proof Let $G = (V,E)$ be a 1-plane graph. Both GREEDY and REVERSE GREEDY behave in the same way on 1-plane graphs: They add uncrossed edges to E' whenever there is one. They remove edges closing a cycle. Finally, they pick a random pair of crossing edges, remove one of them, and move the other to E'. We have to show that this algorithm computes the minimum number of connected components.
Assume that at some point we have computed a set E' that could be extended to an optimal solution M of the cluster minimization problem, that we added $e = \{v,w\}$ and that $E' \cup \{e\}$ cannot be extended to an optimum solution. Thus, M must contain an edge e' that crosses e. We may assume that M does not contain any cycles. If v and w are not in the same connected component in the graph (V,M), then $(M \setminus \{e'\}) \cup \{e\}$ is also crossing-free and induces the same number of components as M. Thus, there is exactly one v-w-path P in M. Since there was no v-w-path in E' at the time when we considered e, there must be an edge e_1 on P that is crossed by an edge $e'_1 = \{v',w'\}$. The edge set $M' := M \setminus \{e_1,e'\} \cup \{e'_1,e\}$ is crossing-free.
If v' and w' are in different connected components of (V,M), then M' induces the same number of connected components as M. Otherwise the unique v'-w'-path P' in M must cross the cycle C induced by P and e. Since e'_1 and e_1 cross, they must stem from different connected components of G. Hence P' crosses C in an edge. Since M is crossing-free, P' crosses C in e. By 1-planarity it follows that P' contains e'. Hence, also in this case, M' induces the same number of connected components as M; contradicting the assumption that $E' \cup \{e\}$ cannot be extended to an optimal solution.
For the runtime, observe that both, the number of edges and the number of crossings of a 1-plane graph with n vertices are in $\mathcal{O}(n)$. Thus all crossings can be computed in $\mathcal{O}(n \log n)$ time. Instead of iteratively applying Step (1), it suffices to first compute a spanning forest of uncrossed edges and then to check each time when we pick a pair of crossing edges whether one of them has its end vertices in different connected components. □

5.2. An Exact Method

Linear programming is a popular tool for solving combinatorial optimization problems. A linear program (LP) consists of real-valued variables $x_1,\dots,x_n$, a target function that is restricted to be linear in the variables (minimize (or maximize) $c_1x_1 + \cdots + c_nx_n$ for some constants $c_1,\dots,c_n$), and a set of linear constraints (for $i = 1,\dots,m$, $a_{i,1}x_1 + \cdots + a_{i,n}x_n \geq b_i$). Linear programs can be solved efficiently [Kar84]. A (mixed-) *integer* linear program (ILP) is a generalization of a linear program where some variables can be restricted to integer values. Often, binary (that is, 0–1) variables are used. This makes it possible to encode NP-hard optimization problems, hence ILPs cannot be solved efficiently in general, but in practice small and medium-sized instances of such problems can often be solved relatively fast. For our problem we could solve each instance of up to 100 points within 1.8 hours, and some instances with 150 points, each within about a day; see Section 7.

We restate CLUSTER MINIMIZATION in a way that is convenient for deriving an ILP formulation. Given an undirected geometric graph $G = (V,E)$, we want to find a smallest set of plane connected subgraphs of G whose vertex sets partition V. As above, we refer to these plane subgraphs as clusters. We can think of the clusters as trees, but we don't forbid cycles. For a vertex $v \in V$, let $N(v)$ denote the neighborhood of v. Set $n = |V|$. We identify the potential clusters with numbers in $C = \{1,2,\dots,n\}$.

Following an idea of Shirabe [Shi05], we define a flow in G and constrain it such that the connected sets of edges that carry positive flow define the clusters. Since we need flow to go either direction along an edge, we introduce the arc set $A = \{uv, vu \colon \{u,v\} \in E\}$. We need the following variables, all of which are binary (except the flow variables). To help intuition, we specify the intended meaning of setting the variables to 1.

$x_{uv} = 1 \Leftrightarrow$ arc $uv \in A$ is in the solution
$a_c = 1 \Leftrightarrow$ cluster $c \in C$ contains at least one vertex
$t_{vc} = 1 \Leftrightarrow$ vertex v is a sink in cluster c
$t_v = 1 \Leftrightarrow$ vertex v is a sink
$s_v = 1 \Leftrightarrow$ vertex v is a source
$f_{uv} \geq 0$ denotes the flow along arc $uv \in A$ (i.e., from u to v)

Then our ILP for minimizing the number of clusters is as follows.

Minimize $\sum_{c \in C} a_c$ subject to the following conditions.

Edge connections are symmetric:

$$x_{uv} = x_{vu} \qquad \text{for } uv \in A$$

For each pair of crossing edges, pick at most one edge:

$$x_e + x_{e'} \leq 1 \qquad \text{for each crossing } (e, e') \in A^2$$

A vertex is a sink if it is the sink for any cluster:

$$t_v = \sum_{c \in C} t_{vc} \leq 1 \qquad \text{for } v \in V$$

Each vertex is either a source or a sink:

$$s_v = 1 - t_v \qquad \text{for } v \in V$$

Each cluster with a sink is counted:

$$\sum_{v \in V} t_{vc} = a_c \leq 1 \qquad \text{for } c \in C$$

Flow can only pass through selected edges:

$$f_{uv} \leq n \cdot x_{uv} \qquad \text{for } uv \in A$$

Net outflow from sources is 1, net outflow from sinks is at most 0:

$$s_v - n \cdot t_v \leq \sum_{u \in N(v)} (f_{vu} - f_{uv}) \leq s_v \qquad \text{for } v \in V$$

The last constraint forces the net outflow $\sum_{u \in N(v)} (f_{vu} - f_{uv})$ for each vertex v to be 1 if v is a source (that is, if $s_v = 1$). This is due to the fact that $s_v = 1$ implies that the left-hand side and the right-hand side are both 1. Otherwise, v is a sink (that is, $s_v = 0$ and $t_v = 1$). Then the net outflow is bounded from below by $1 - n$ and from above by 0. The conditions of the ILP can be fulfilled if and only if $\{\{u, v\} \in E : x_{uv} = 1\}$ is crossing-free and induces at most as many connected components as there are sinks. In an optimal solution there is exactly one sink in each connected component.

6. Case Study

In this section, we present and compare results obtained with different visualization methods, including our own methods and methods from the literature. For all results computed with our own methods that are presented in this section, we used a β-skeleton as underlying neighborhood graph and the greedy heuristic to compute the clustering. Our aim is to shed light on how the choice of β and the choice of the method for displaying the resulting clusters affect the quality of the visualization with respect to Design Goals 1–4.

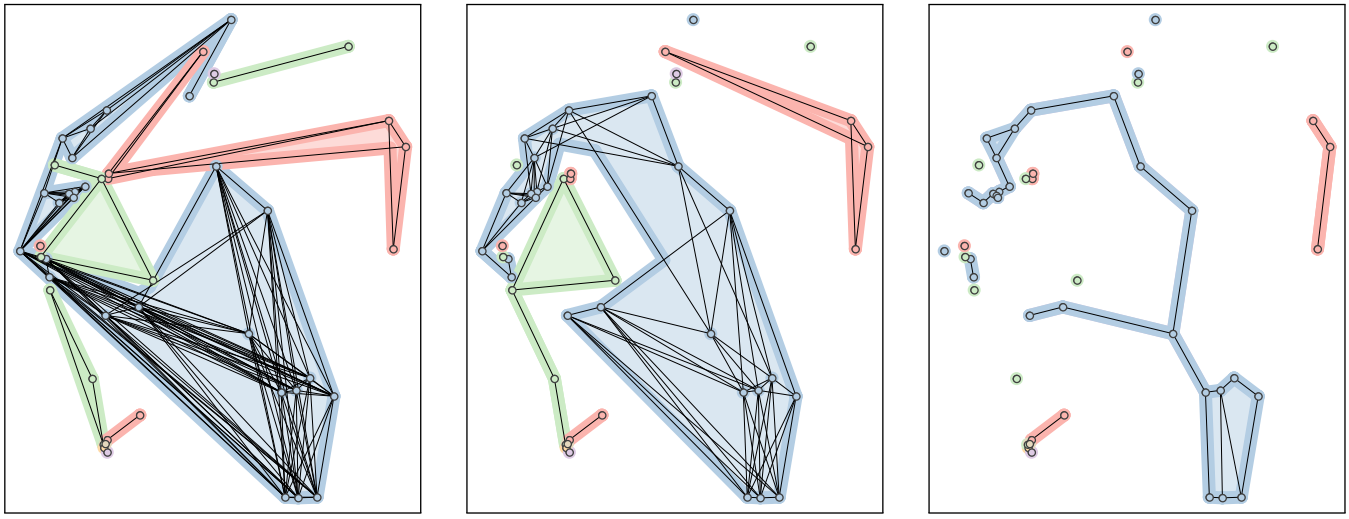
We tested our methods for three different data sets, which we refer to as **UBN**, **OSM**, and **NYC**.

UBN is a set of 78 points representing facilities of the University of Bonn, Germany, where each facility belongs to one of seven departments defining the categories. A small part of this set of points is shown in Fig. 1. After retrieving the points [Goo], we classified them by department membership and excluded points that we could not assign to any department. We consider this data set interesting since points of the same category tend to form local clusters, simply because facilities of the same department are located close to each other. Nevertheless, the point sets of different categories are interwoven spatially, making it challenging to find a crossing-free graph of few connected components spanning the point set.

We computed two different visualizations for UBN, which are shown in parts in Fig. 1 and at full extent in the supplementary material. The first visualization is simply using a Voronoi diagram of the point set to partition the plane into regions of different colors; see Fig. 1 (left). This is a popular approach in Geographic Information Science to generate polygonal maps from categorical point data [GNY96, AP08]. Although most facilities of the Department of Agriculture (brown) and the Department of Mathematics and Natural Sciences (yellow) are located on or close to the main university campus, both point sets are split over multiple regions. Only the Department of Protestant Theology (red) with only one building and the Department of Law (orange) with two buildings constitute contiguous regions. In contrast, when using our greedy algorithm to select edges from a β-skeleton with $\beta = 0.5$ and additional intra-cluster edges, four of the seven departments are represented with a single cluster; see Fig. 1 (right). Approximating a line Voronoi diagram by sampling points on the selected edges yields a partition of the plane into relatively compact regions corresponding to the clusters, which can be thought of as a political map or categorical coverage map; see the transparent, colored areas in Fig. 1 (right). A disadvantage of the Voronoi-based methods is that large empty regions are colored, which might seem incomprehensible to the user. Moreover, although the colored regions are displayed transparent, they substantially reduce the readability of the content in the background. Therefore, we consider the method that fills only the areas enclosed by the displayed edges as a good alternative; see the opaque areas in Fig. 1 (right). A drawback of this method is, however, that singletons and other clusters that do not enclose any area are hardly visible. This drawback can be compensated by combining the polygon-based visualization with the line-Voronoi method (as in Fig. 1 (right)) or displaying colored halos around singletons and edges, as we will show on two other data sets next.

OSM is a set of 16320 points of interest (POIs) that we extracted from a file of OpenStreetMap data provided for download [GO]. It has a similar spatial extent as UBN covering the whole city of Bonn. These points belong to 123 different categories, of which some constitute local clusters (e.g., POIs of category "clothes" in shopping centers) while others do not (e.g., POIs of category "bakery"). We used this data set in particular to draw smaller samples from it for the experiments presented in Sect. 7, which we conducted to analyze the running times of our methods and their performance in terms of quality, for instances of different sizes. In the following, we use this data set to study the influence of β.

Figure 7 shows visualizations we obtained by applying our method with different values for β to a subset of points from OSM. For the same data set, Fig. 8 shows the influence of β on the number of clusters. (The same figure for the other data sets can be found in the supplemental materials.) While with smaller β the number of clusters decreases substantially, it appears that the choice of REVERSE GREEDY or GREEDY has only a minor influence. In the special case $\beta = 0$, the β-skeleton is the complete graph. Accordingly, choosing it leads to few and relatively large clusters, satisfying Design Goal 4; see Fig. 7a. On the other hand, some clusters are connected only via very long edges, contradicting Design Goal 3. We also note that a cluster might completely enclose another cluster—a situation that may be difficult to comprehend. Since this can also be interpreted as a hole in the larger polygon, we do not explicitly forbid nested clusters, but allow the user to specify a threshold θ



(a) $\beta = 0.0$*; 11 clusters* **(b)** $\beta = 0.5$*; 15 clusters* **(c)** $\beta = 1.0$*; 22 clusters*

Figure 7: *Greedy solutions for different values of* β *on a section of the OSM data. In postprocessing, we added intra-cluster edges from the original* β*-skeleton. We added only those edges that would not cross edges of other clusters or enclose other clusters. We then applied the polygonal visualization style to the resulting set of edges.*

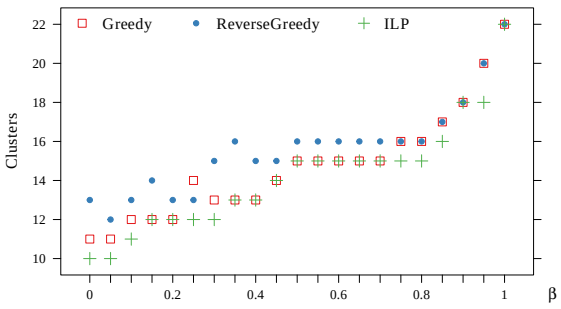
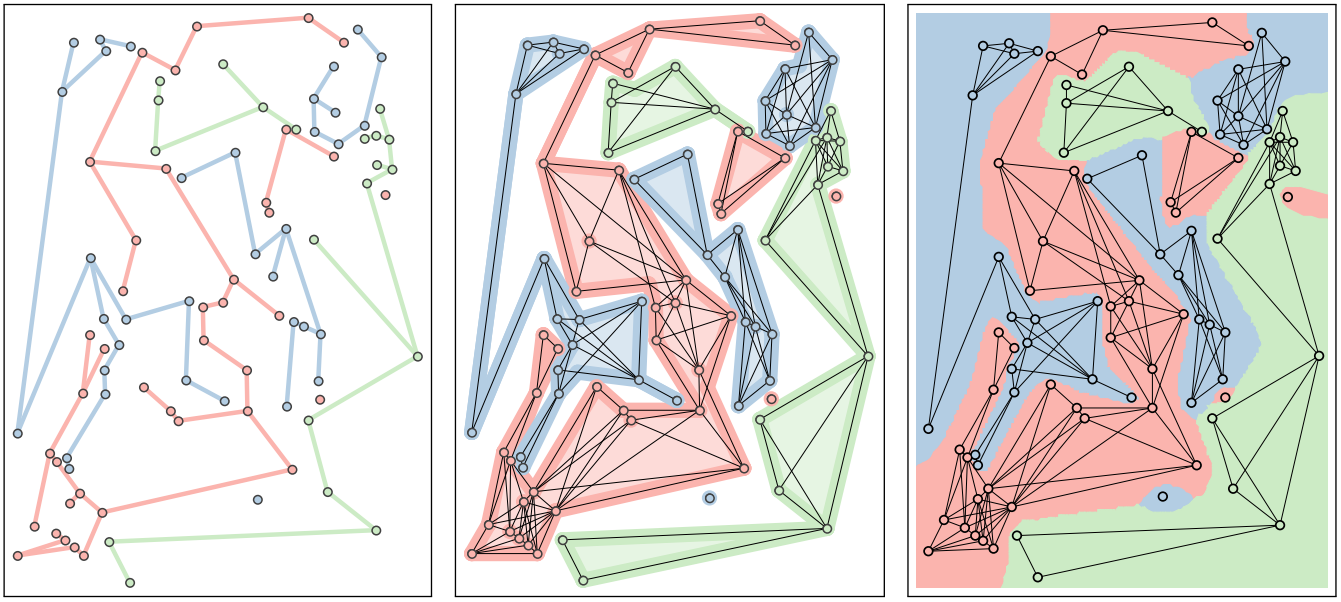


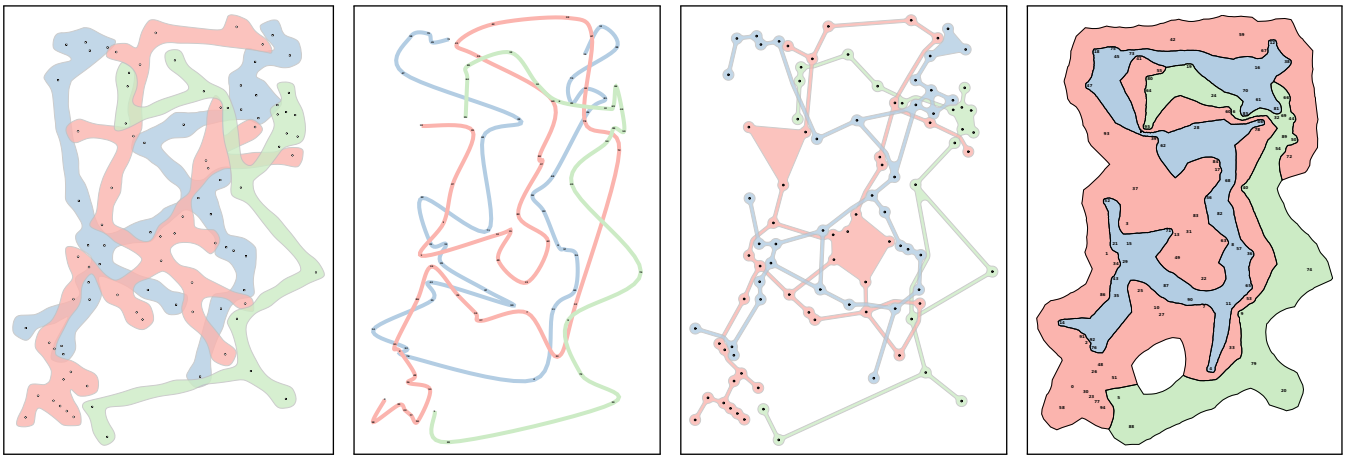
Figure 8: *Influence of* β *on the cluster number produced by the ILP,* GREEDY*, and* REVERSE GREEDY *for the instance in Fig. 7.*

that forbids the algorithm to enclose clusters of size at most θ. In our experiments we observed that setting $\theta = 1$ worked well. For $\beta = 1$, which is the other extreme, we obtain the Gabriel graph, which is crossing-free. The partition of the points into clusters is thus determined already with the proximity graph, while the edge augmentation step may add some additional intra-cluster edges; see Fig. 7c. Since we obtain many small clusters or even singletons, the method fails entirely to satisfy Design Goal 4. The solution for $\beta = 0.5$, which is shown in Fig. 7b, can be considered a good trade-off between Design Goals 3 and 4 since the clusters are relatively large and the edges connect points that are relatively near to each other. Design Goals 1 and 2 are reached for all β.

NYC is a set of 96 points that are located in Manhattan, New York, each belonging to one of the three categories "hotel", "subway station", and "clinic". We use this data set primarily since it has been used as a benchmark before. In particular, Fig. 9 shows three different visualizations generated with our ClusterSets method as well as visualizations of Bubble Sets [CPC09], LineSets [AHRRC11], KelpFusion [MHRS*13], and MapSets [EHKP15]. The four methods from the literature shown in Figs. 9d–9g ensure that all points of the same category are displayed with a connected shape. Generally, this is difficult without tolerating crossing lines or overlapping areas; only MapSets, by design, achieves crossing-free, yet rather complex cluster shapes; see Fig. 9g. Overlaps and crossings can be particularly problematic if happening near set elements as the shape intersection may create ambiguities in the category memberships of enclosed or nearby points; see Figs. 9d–9f. It is not clear, however, whether the global connectivity per category justifies the resulting line crossings, some of which form acute crossing angles or occur close to data points. While MapSets does not create any overlaps of the cluster shapes, the connectivity requirement comes at the cost of highly complex shape boundaries and non-compact clusters, see, e.g., the red cluster wrapping around the blue one and claiming the empty area on the left-hand side in Fig. 9g. Unlike our method, the four systems have not been designed to separate local clusters of the same category, which limits their range of applicability. With respect to our own visualizations, we argue that the tree representation (Fig. 9a) is in a sense similar to KelpFusion and LineSets since it shows lines that connect points of the same category without giving a good idea of local clusters. Local clusters become more apparent, however, with the display of additional intra-cluster edges and polygons enclosing the clusters (Fig. 9b) or Voronoi regions partitioning the plane (Fig. 9c). The representation based on the line Voronoi diagram works particularly well in this example, since the points are almost uniformly distributed over the extent of the data set. Recall, however, that the polygon representation (Fig. 9b) should be chosen if a data set (such as UBN) contains large regions without any points.



(a) *ClusterSets: Tree representation* **(b)** *ClusterSets: Polygon representation* **(c)** *ClusterSets: Line-Voronoi representation*



(d) *Bubble Sets [CPC09]* **(e)** *LineSets [AHRRC11]* **(f)** *KelpFusion [MHRS*13]* **(g)** *MapSets [EHKP15]*

Figure 9: *Comparison of different visualizations for the NYC data. We digitized the Bubble Sets representation from Figure 9 of the original paper [CPC09]. The LineSets and MapSets representations were produced at* `http://gmap.cs.arizona.edu`.

7. Quantitative Experiments

We conducted experiments to compare the running time and quality of the GREEDY and REVERSE GREEDY heuristic with the ILP for different values of β. In particular, we wanted to measure if one of the two heuristics performs better in practice and how close the solutions are to the optimum.

Experimental Setup We created 45 data sets based on the OSM data as follows. We divided the map that contains all points of interest uniformly into nine squares and computed the center of each square. Then, for $n = 50, 100, 150, 200, 250$, we computed the n closest points of interest for each center. This way, we created 9 different data sets for each n. We restricted β to values in the range from 0.5 to 0.9 in steps of 0.05, as larger values created too many clusters (the β-skeleton is too sparse) and smaller values do not represent geometric closeness well (the β-skeleton is too dense; recall that for $\beta = 0$ all possible edges are in the β-skeleton). We computed the solutions of our greedy heuristics and the ILP for all data sets for these values of β .

We ran our experiments on a server with an Intel Xeon X5550 processor with 16 cores of 2.67 GHz. The server runs under Ubuntu 20.04 and has 16 GB of memory. Our heuristics were implemented in Java (openjdk 11.0.9.1) and the ILP was implemented in IBM ILOG CPLEX Optimization Studio 12.9.0.0. The code is available at `https://github.com/JakobGeiger/ClusterSets`.

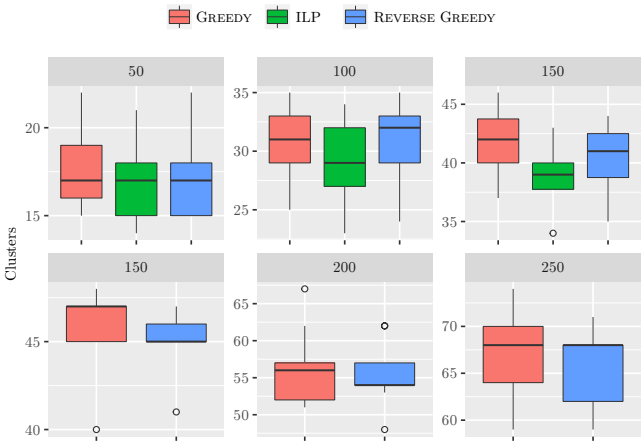


Figure 10: *Comparison of solutions for* $\beta = 0.5$. *The y-axis counts the number of clusters in the solutions. In the top row we show the instances for which we found an optimum solution with the ILP; in the bottom row we show the remaining instances.*

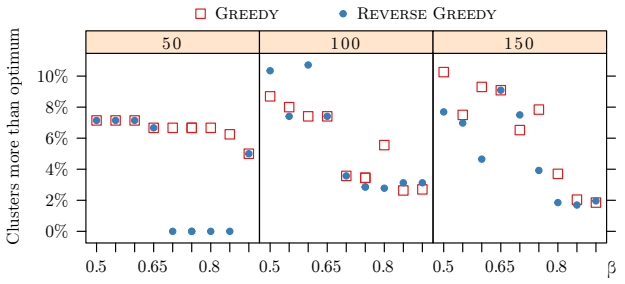


Figure 11: *Comparison of solutions by our greedy heuristics with the optimum. The y-axis shows the additional number of clusters (in percent) computed by the heuristics compared to the optimum for each* β, *grouped by number of points.*

Results Both heuristics were able to compute a solution within less than 50 ms for all instances. On the other hand, the ILP was considerably slower. For instances with 50 points it required up to 11 seconds, for instances with 100 points it required up to 1.8 hours, and for instances with 150 points it required up to 2 days. Hence, we did not run the ILP for instances with 200 and 250 points, and only for some of the instances with 150 points.

In Fig. 10, we analyze the number of clusters produced by the heuristics and the ILP grouped by the number of points of an instance for $\beta = 0.5$ as a whiskers plot. The lower hinge is the 25th percentile, the middle hinge is the 50th percentile (the median), and the upper hinge is the 75th percentile. We display the results for $\beta = 0.5$ because the differences between the algorithms are larger for this choice of β than for larger ones. For all instances, the number of clusters produced by the two heuristics were close. Out of the 405 created instances, Greedy gave the better solution 66 times, Reverse Greedy performed better 126 times, and they produced the same number of clusters 213 times. The same plot for the other values of β is available in the supplemental materials.

Surprisingly, the solutions of our heuristics were close to the optimum: they produced at most 10% more clusters than the optimum solution for $\beta = 0.5$, at most 7.5% more clusters for $\beta = 0.75$, and at most 5% more clusters for $\beta = 0.9$; see Fig. 11.

8. Discussion and Limitations

Finally, we discuss limitations and improvements of our approach.

Both heuristics consider only the number of crossings of an edge to decide which edges to pick or to remove. In general, close points should be more likely to be in the same cluster than points that are far away from each other. To this end, one could try to also minimize the total edge length of the solution, for example by breaking ties in the greedy approach by preferring shorter edges.

We have considered only data in which every point belongs to exactly one category. In practice, it could happen that some points belong to more than one category. Our heuristics do not easily extend to cover this more general case, and probably some more involved approaches are necessary to solve it.

In the Edge Augmentation step, we added edges to the spanning tree in an arbitrary order. It might be possible to obtain better results by using heuristic approaches similar to Greedy and Reverse Greedy to maximize the number of edges that are added.

When clustering categorized geographic data, crossings between edges connecting vertices of the same category could also be used to connect clusters. For 1-plane graphs, we discussed replacing such intra-category crossings by 4-stars (see Theorem 1). It is not clear, however, how to generalize this approach if edges are involved in both intra- and inter-category crossings.

Cluster Minimization is also NP-hard in the case that all vertices belong to the same category since Jansen and Woeginger [JW93] showed that it is NP-hard to decide whether a given geometric graph contains a crossing-free spanning tree. While this case is not so interesting for our clustering application, we can show, using a simple trick [KSSW07], that "monochromatic" Cluster Minimization is even hard to approximate.

Consider the geometric graph that Jansen and Woeginger [JW93] construct to show that crossing-free spanning tree is NP-hard. If we place k disjoint copies of this graph along the x-axis and connect the rightmost vertex of one copy with the leftmost vertex of the next copy, then we get a new geometric graph for which it is NP-hard to distinguish between the case that it has a plane spanning tree (i.e., one cluster) and the case that any plane subgraph has at least $k+1$ connected components (i.e., many clusters).

Acknowledgments We thank the organizers of Dagstuhl seminar 19192, where this work originated. Among the participants, we particularly thank Hugo Akitaya for discussions. We thank Joachim Spoerhase for simplifying our NP-hardness argument. We thank Wouter Meulemans for running his KelpFusion implementation on several instances for us. We thank Sergey Pupyrev for adjusting his code to produce Fig. 9g for us.

References

[AHRRC11] ALPER B., HENRY RICHE N., RAMOS G., CZERWINSKI M.: Design study of LineSets, a novel set visualization technique. *IEEE Trans. Visualization and Computer Graphics 17*, 12 (2011), 2259–2267. doi:10.1109/TVCG.2011.186. 3, 8, 9

[AMA*16] ALSALLAKH B., MICALLEF L., AIGNER W., HAUSER H., MIKSCH S., RODGERS P. J.: The state-of-the-art of set visualization. *Computer Graphics Forum 35*, 1 (2016), 234–260. doi:10.1111/cgf.12722. 3

[AOA19] AKSAC A., ÖZYER T., ALHAJJ R.: CutESC: Cutting edge spatial clustering technique based on proximity graphs. *Pattern Recognition 96* (2019), 106948. doi:10.1016/j.patcog.2019.06.014. 3

[AP08] ABELLANAS M., PALOP B.: Urban data visualization with Voronoi diagrams. In *Proc. International Conference Computational Science and Its Applications (ICCSA), Part I* (2008), Gervasi O., Murgante B., Laganà A., Taniar D., Mun Y., Gavrilova M. L., (Eds.), vol. 5072 of *Lecture Notes in Computer Science*, Springer, pp. 126–136. doi:10.1007/978-3-540-69839-5_10. 7

[Ber11] BERTIN J.: *Semiology of Graphics: Diagrams, Networks, Maps*. ESRI Press, 2011. 3

[BO79] BENTLEY J. L., OTTMANN T. A.: Algorithms for reporting and counting geometric intersections. *IEEE Trans. Computers C-28*, 9 (1979), 643–647. doi:10.1109/TC.1979.1675432. 5

[BP02] BECK D. M., PALMER S. E.: Top-down influences on perceptual grouping. *Journal of Experimental Psychology: Human Perception and Performance 28*, 5 (2002), 1071–1084. doi:10.1037/0096-1523.28.5.1071. 3

[CLS*19] CHAKRABARTY D., LEE Y. T., SIDFORD A., SINGLA S., WAI WONG S. C.: Faster matroid intersection. In *Proc. 60th Annual Symposium on Foundations of Computer Science (FOCS)* (2019), IEEE, pp. 1146–1168. doi:10.1109/FOCS.2019.00072. 6

[CPC09] COLLINS C., PENN G., CARPENDALE S.: Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Visualization and Computer Graphics 15*, 6 (2009), 1009–1016. doi:10.1109/TVCG.2009.122. 2, 3, 8, 9

[CvGM*19] CASTERMANS T., VAN GARDEREN M., MEULEMANS W., NÖLLENBURG M., YUAN X.: Short plane supports for spatial hypergraphs. *J. Graph Algorithms Appl. 23*, 3 (2019), 463–498. doi:10.7155/jgaa.00499. 3

[dBCvKO08] DE BERG M., CHEONG O., VAN KREVELD M., OVERMARS M.: *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, Berlin, 2008. 2, 4

[DvKSW12] DINKLA K., VAN KREVELD M., SPECKMANN B., WESTENBERG M. A.: Kelp diagrams: Point set membership visualization. *Computer Graphics Forum 31*, 3 (2012), 875–884. doi:10.1111/j.1467-8659.2012.03080.x. 3

[EHKP15] EFRAT A., HU Y., KOBOUROV S. G., PUPYREV S.: MapSets: Visualizing embedded and clustered graphs. *J. Graph Algorithms Appl. 19*, 2 (2015), 571–593. doi:10.7155/jgaa.00364. 3, 8, 9

[EKASX96] ESTER M., KRIEGEL H.-P., A SANDER J., XU X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (SIGKDD)* (1996), pp. 226–231. doi:10.5555/3001460.3001507. 3

[GB20] GRÖBE M., BURGHARDT D.: Micro diagrams: Visualization of categorical point data from location-based social media. *Cartography and Geographic Information Science 47*, 4 (2020), 305–320. doi:10.1080/15230406.2020.1733438. 3

[GHK10] GANSNER E. R., HU Y., KOBOUROV S. G.: GMap: Visualizing graphs and clusters as maps. In *Proc. Pacific Visualization Symposium (PacificVis)* (2010), IEEE, pp. 201–208. doi:10.1109/PACIFICVIS.2010.5429590. 3

[GHvK02] GUDMUNDSSON J., HAMMAR M., VAN KREVELD M.: Higher order Delaunay triangulations. *Comput. Geom. Theory Appl. 23* (2002), 85–98. doi:10.1016/S0925-7721(01)00027-X. 2, 4

[GNY96] GOLD C. M., NANTEL J., YANG W.: Outside-in: an alternative approach to forest map digitizing. *International Journal of Geographical Information Systems 10*, 3 (1996), 291–310. doi:10.1080/02693799608902080. 7

[GO] GEOFABRIK GMBH, OPENSTREETMAP CONTRIBUTORS: Regierungsbezirk Köln. URL: https://download.geofabrik.de/europe/germany/nordrhein-westfalen/koeln-regbez-latest-free.shp.zip. 7

[Goo] GOOGLE MAPS: Standorte der Universität Bonn. URL: https://www.google.com/maps/d/viewer?mid=1wZ53XzFZeQLRtUuz0FMsH3_GbANYrfQq. 7

[JW93] JANSEN K., WOEGINGER G. J.: The complexity of detecting crossingfree configurations in the plane. *BIT 33* (1993), 580–595. doi:10.1007/BF01990536. 4, 10

[Kar84] KARMARKAR N.: A new polynomial time algorithm for linear programming. *Combinatorica 4*, 4 (1984), 373–395. doi:10.1007/BF02579150. 6

[KN90] KRATOCHVÍL J., NEŠETŘIL J.: Independent set and clique problems in intersection-defined classes of graphs. *Comment. Math. Univ. Carolinae 31*, 1 (1990), 85–93. URL: https://gdz.sub.uni-goettingen.de/id/PPN316342866_0031?tify={"pages":[89]}. 2, 4

[Kra09] KRAAK M.-J.: Geovisualization. In *International Encyclopedia of Human Geography*, Kitchin R., Thrift N., (Eds.). Elsevier, Oxford, 2009, pp. 468–480. doi:10.1016/B978-008044910-4.00033-X. 2

[KSSW07] KNAUER C., SCHRAMM É., SPILLNER A., WOLFF A.: Configurations with few crossings in topological graphs. *Comput. Geom. Theory Appl. 37*, 2 (2007), 104–114. doi:10.1016/j.comgeo.2006.06.001. 10

[KV18] KORTE B., VYGEN J.: *Combinatorial Optimization: Theory and Algorithms*, 6th ed., vol. 21 of *Algorithms and Combinatorics*. Springer-Verlag Berlin Heidelberg, 2018. doi:10.1007/978-3-662-56039-6. 6

[MHRS*13] MEULEMANS W., HENRY RICHE N., SPECKMANN B., ALPER B., DWYER T.: KelpFusion: A hybrid set visualization technique. *IEEE Trans. Visualization and Computer Graphics 19*, 11 (2013), 1846–1858. doi:10.1109/TVCG.2013.76. 3, 8, 9

[MS80] MATULA D. W., SOKAL R. R.: Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis 12*, 3 (1980), 205–222. doi:10.1111/j.1538-4632.1980.tb00031.x. 4

[Shi05] SHIRABE T.: A model of contiguity for spatial unit allocation. *Geographical Analysis 37*, 1 (2005), 2–16. doi:10.1111/j.1538-4632.2005.00605.x. 6

[Zah71] ZAHN C.: Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Computers C-20*, 1 (1971), 68–86. doi:10.1109/T-C.1971.223083. 3