DGK Ausschuss Geodäsie (DGK)
der Bayerischen Akademie der Wissenschaften

Reihe C Dissertationen Heft Nr. 971

Peter Rottmann

Map Generalization and Set Visualization Through Spatial Unit Allocation

München 2025

Verlag der Bayerischen Akademie der Wissenschaften, München

ISSN 0065-5325 ISBN 978-3-7696-5383-0

DGK Ausschuss Geodäsie (DGK)
der Bayerischen Akademie der Wissenschaften

Reihe C Dissertationen Heft Nr. 971

Map Generalization and Set Visualization Through Spatial Unit Allocation

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaft (Dr.-Ing.)
der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Peter Rottmann
aus Bonn

München 2025

Verlag der Bayerischen Akademie der Wissenschaften, München

ISSN 0065-5325 ISBN 978-3-7696-5383-0

Adresse des Ausschusses Geodäsie (DGK)
der Bayerischen Akademie der Wissenschaften:

Ausschuss Geodäsie (DGK) der Bayerischen Akademie der Wissenschaften
Alfons-Goppel-Straße 11 ● D – 80 539 München
Telefon +49 – 89 – 23 031 1113 ● Telefax +49 – 89 – 23 031 - 1283 / - 1100
e-mail post@dgk.badw.de ● http://www.dgk.badw.de

Referent: Prof. Dr. Jan-Henrik Haunert, University of Bonn, Germany

Korreferent: Prof. Dr. Daniel Archambault, Newcastle University, United Kingdom

Tag der mündlichen Prüfung: 28.10.2025

Angefertigt mit Genehmigung der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät der Universität Bonn

Diese Dissertation ist auf dem Server des Ausschusses Geodäsie (DGK)
der Bayerischen Akademie der Wissenschaften unter <http://dgk.badw.de/>
sowie auf dem Server der Rheinischen Friedrich-Wilhelms-Universität Bonn unter
<https://doi.org/10.48565/bonndoc-706> elektronisch publiziert

---

© 2025 Ausschuss Geodäsie (DGK) der Bayerischen Akademie der Wissenschaften, München

Alle Rechte vorbehalten. Ohne Genehmigung der Herausgeber ist es auch nicht gestattet,
die Veröffentlichung oder Teile daraus zu vervielfältigen.

ISSN 0065-5325 ISBN 978-3-7696-5383-0

Abstract

Data visualization is an important aspect of making complex data accessible to a wide audience. Capturing key information in visualized data is critical for users to understand the data and make informed decisions. On the one hand, the data may contain geospatial information, which is often visualized using maps. When navigating maps that visualize spatial data, changing the zoom level is a common strategy for controlling the level of detail. Reducing the level of detail of a map is a key challenge in cartography and can be addressed with map generalization. These changes in level of detail while zooming should be monotonous. On the other hand, the visualization of abstract data, such as set systems, is a well-established field of research in information visualization. Set systems contain information about set elements and their relationships, e.g. working groups and their research projects within a faculty. Set visualization have proven to be a powerful tool for presenting large amounts of data to a user in a compact way. Seeing the interactions between sets in a visualization is easier to understand than reading a description of those interactions. However, both types of data, geospatial and abstract, can become overwhelming for the user if every detail is visualized.

In this thesis, we present optimization approaches for creating generalizations for both geospatial data and set systems. In our algorithms, we adapt some approaches from the research field of spatial unit allocation. When generalizing geospatial data, we aggregate small polygons into larger areas, aiming for a set of representative output polygons. We use our approach to compute a set of solutions that contains an optimal solution for each value of the parameter which controls the level of generalization. For set systems, we present the first approach that is able to determine whether a given set system can be completely displayed as an Euler diagram that satisfies certain *wellformedness conditions*. If a complete visualization is not possible or required, we aim to create a visualization that either shows as much information as possible or improves readability by losing minimal information. Such a resulting set system can be visualized with our novel visualization technique called *MosaicSets*. The novel technique builds on an underlying grid and creates a mosaic-like visualization of the set system. We optimize each set to form a compact region to improve the readability of the diagram. For all our approaches, we provide theoretical analysis and evaluate the performance of our algorithms on numerous real-world datasets.

Kurzfassung

Datenvisualisierung ist ein wichtiger Aspekt, um komplexe Daten einem breiten Publikum zugänglich zu machen. Dabei ist die Erfassung von Schlüsselinformationen entscheidend, um die Daten zu verstehen und fundierte Entscheidungen treffen zu können. Einerseits können die Daten raumbezogene Informationen enthalten, die oft als Karten dargestellt werden. Bei der Handhabung dieser Karten ist die Änderung der Zoomstufe eine gängige Vorgehensweise zur Steuerung des Detailgrads. Diese Änderungen sollten monoton sein. Die Verringerung des Detaillierungsgrads ist eine zentrale Herausforderung in der Kartografie und kann durch Generalisierung der Karten gelöst werden. Andererseits ist die Visualisierung abstrakter Daten, wie z.B. von Mengensystemen, ein etabliertes Forschungsgebiet in der Informationsvisualisierung. Mengensysteme enthalten Informationen über Mengenelemente und deren Beziehungen, z.B. Arbeitsgruppen und deren gemeinsame Forschungsprojekte. Die Visualisierung von Mengen hat sich als ein leistungsfähiges Werkzeug herausgestellt, um dem Nutzer große Datenmengen in kompakter Form darzustellen. Visualisierte Beziehungen zwischen Mengen lassen sich leichter verstehen als textuelle Beschreibungen. Allerdings vereint raumbezogene und abstrakte Daten, dass es für den Nutzer überfordernd sein kann, wenn jedes Detail dargestellt wird.

In dieser Arbeit werden Optimierungsansätze für die Erstellung von Generalisierungen für Geodaten und für Mengensysteme vorgestellt. In den Algorithmen werden Ansätze aus dem Forschungsbereich der Zuweisung räumlicher Flächeneinheiten adaptiert. Bei der Generalisierung von Geodaten werden kleine Polygone zu größeren Flächen zusammengefasst. Dieser Algorithmus wird verwendet, um eine Lösungsmenge zu berechnen, welche eine optimale Lösung für jeden Parameterwert, welcher den Grad der Generalisierung bestimmt, enthält. Für Mengensysteme wird der erste Ansatz vorgestellt, welcher bestimmt, ob ein gegebenes Mengensystem unter Berücksichtigung von *Wohlgeformtheitsbedingungen* vollständig als Euler-Diagramm dargestellt werden kann. Wenn eine vollständige Darstellung nicht möglich oder erforderlich ist, ist es das Ziel, eine Darstellung zu erzeugen, die entweder so viele Informationen wie möglich zeigt oder die Lesbarkeit durch das Entfernen minimaler Informationen verbessert. Ein solches Mengensystem kann mit der neuen eingeführten Visualisierungstechnik *Mosaic-Sets* visualisiert werden. Diese baut auf einem zugrunde liegenden Raster auf und erzeugt eine mosaikartige Visualisierung des Mengensystems. Jede Menge wird so optimiert, dass sie eine kompakte Region bildet, um die Lesbarkeit des Diagramms zu verbessern. Für alle gezeigten Ansätze werden theoretische Analysen erstellt und die Leistung der Algorithmen anhand zahlreicher realer Datensätze bewertet.

Acknowledgments

First and foremost, I would like to thank my supervisor Jan-Henrik Haunert for his support and guidance during my research. I am grateful for the opportunity to work on this exciting topic and for the valuable ideas, feedback and insights you provided along the way. I would also like to thank for the possibilities to present my work at various conferences and workshops, which helped me to improve my research and connect with other researchers in the field. Attending conferences in the USA and Denmark was a great experience.

Many thanks to all my co-authors for their collaboration and support on various projects. Your experience in various fields of research has been invaluable to me, and I have learned a lot from our discussions and collaborations. I want to thank Peter Rodgers and Daniel Archambault for their support on Euler diagrams and for providing source code of their previous work, which was a great help for my research.

I would like to express my gratitude to the colleagues of the research group for their support and encouragement. I am not only grateful for the scientific discussions during meetings, but also for joint activities outside of work, which made my time at the university even more enjoyable. Special thanks go to Annika and Sven, who were not only great colleagues and collaborators, but also proofreaders of my thesis. I would also like to thank Julius for our various on- and off-topic discussions, for being a great office mate and having great whiteboard sessions.

Finally, I would like to express my gratitude to my wife, Franziska, and my family for their unwavering support and encouragement. Their belief in me and their understanding have been invaluable. I am grateful for their patience and support, which have helped me to stay focused and motivated throughout this journey.

Contents

1 Introduction **1**
1.1 Contributions 5

2 State of the Art in Map Generalization and Set Visualization **9**
2.1 Map Generalization 9
2.1.1 Aggregation and Amalgamation 11
2.1.2 Continuous Generalization 13
2.2 Set Visualization 14

3 Algorithmic Fundamentals **19**
3.1 Graph Theory 19
3.1.1 Graph Fundamentals 19
3.1.2 Hypergraph Drawings and Planar Supports 24
3.2 Integer Linear Programming 26
3.3 Spatial Unit Allocation 28
3.3.1 Districting 30
3.4 Graph Cuts 31

4 Map Generalization Using Graph Cuts **35**
4.1 Introduction 36
4.2 Methodology 38
4.2.1 Graph Cut 39
4.2.2 Hierarchical Structure of Solutions for Multiple Parameter Values 41
4.3 Experiments 45
4.3.1 Choosing the Parameter λ with the Help of Reference Solutions 51
4.3.2 Edge-Aligned Polygons 52
4.3.3 Comparison to α-shapes 54
4.4 Conclusion 55

5 Grid-Based Euler Diagrams **57**
5.1 Introduction 58
5.2 Contextual Background 60
5.2.1 Geographic Information Visualization Techniques 60
5.2.2 Maps as a Metaphor 62
5.3 Towards a Formalization of MosaicSets 62
5.3.1 Design Decisions 62

5.3.2 Formal Problem Definition 64
5.3.3 Computational Complexity 65
5.4 An Approach Based on Integer Linear Programming 66
5.4.1 A Basic Integer Linear Program 66
5.4.2 Compactness of Regions Representing Sets 68
5.4.3 Relaxing the Contiguity Requirement 70
5.4.4 An Integer Linear Program with Fewer Variables 71
5.5 Rendering 71
5.6 Evaluation 73
5.6.1 Expert Interviews 73
5.6.2 Tasks for Set Visualizations 76
5.6.3 Experimental Setup 76
5.6.4 Number of Overlay Sets 77
5.6.5 Running Time 78
5.6.6 Assessing the Compactness 79
5.7 Conclusion 79

6 Generating Euler Diagrams Through Combinatorial Optimization 81
6.1 Introduction 82
6.2 Preliminaries and Basic Problem 83
6.3 Workflow 84
6.4 Methodology 86
6.4.1 An Integer Linear Program 87
6.4.2 Heuristic Approach 89
6.5 Experiments 93
6.6 Conclusion 98

7 Conclusion and Future Work 101
7.1 Summary of the Contributions 101
7.2 Future Work 102

Appendices 131

A Supplemental Mosaic Sets 131
A.1 Additional Figures 131
A.2 Task Taxonomy 132
A.3 Expert Interviews 132

B Supplemental Euler Diagrams 149
B.1 Euler Diagram Simplification 149
B.2 Parameter Influence 149

Chapter 1

Introduction

Visualizing spatial as well as abstract information is important to enhance the human understanding of complex data. When it comes to spatial data, maps are a common visualization technique. In some cases, it may be sufficient to display all available data as a map. However, in many use cases, the complete data set is too detailed to create a map for a user. Such use cases, where maps are too detailed, can be background maps for car navigation systems, maps on mobile phones, or small-scale topographic maps. As these examples illustrate, the level-of-detail of a map must be appropriate with respect to the scale and application scenario of the map.

Map generalization is the process of deriving a less detailed map from a given map. In cartography, a well-established approach to avoid cluttered maps is the use of map generalization. It comprises multiple sub-tasks such as the selection, simplification, aggregation, and displacement of objects (Hake et al., 2002). An example of map generalization is given in Figure 1.1. In the displayed example,

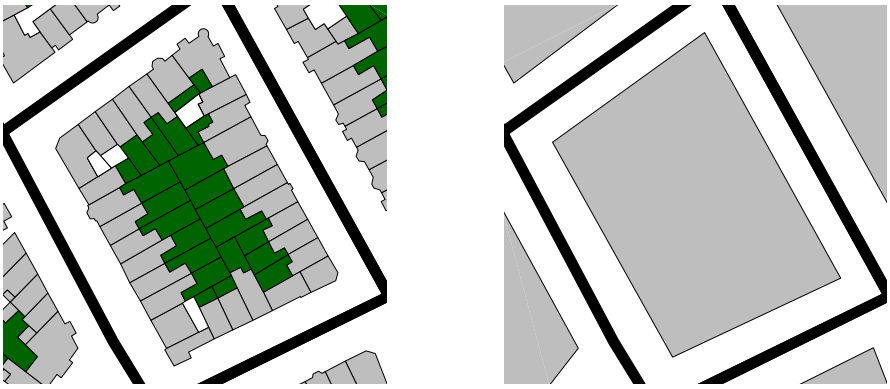
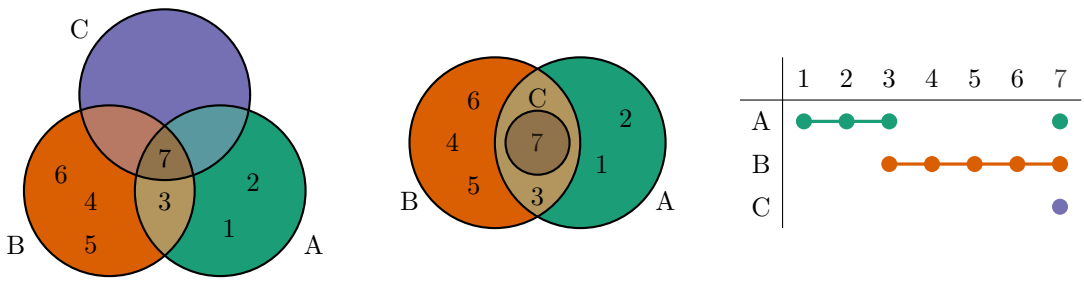


Figure 1.1: An example of map generalization. The left image shows the original map with all building footprints in gray and garden areas in green. The right image shows the generalized map with settlement areas (gray), where garden areas are included in the settlement areas. The black lines represent streets and are not part of the aggregation.

the building footprints of the given map are aggregated into settlement areas and the resulting polygon is simplified. In this thesis, we address the task of detecting polygon groups (i.e., spatial clusters of polygons) and aggregating each group into a single polygon, which we simply refer to as *aggregation of polygons*. This task is relevant, e.g., to derive settlement areas from mutually disjoint building footprints. In this case, aggregation and simplification of polygons is performed. However, it is possible to combine additional generalization operators to further improve the quality of the map, e.g., to reduce clutter in small-scale maps by selecting only large polygons for display on small-scale maps. Selecting the largest polygons is equivalent to showing only the most important and largest cities on a small-scale map.

While maps are created for geometric data, information visualization involves the representation of abstract data. The goal of information visualization is to create visual representations of complex data sets to help users to understand the data. Such visualizations can be visualizations of hierarchies, such as organizational data, or set systems, such as Venn diagrams or flowcharts. We show an example of a Venn diagram in Figure 1.2a. Similar to map generalization, the goal of information visualization is to create a readable and understandable representation of the data. If the data is too complex, users will have trouble focusing on the important information, which will hinder their ability to make effective use of the data. Hence, the user's attention should be directed to the most important information. Therefore, reducing the data to the most important data points is critical before creating a visualization. In this thesis, we focus on the visualization of set systems with abstract data. A set system is a collection of sets, where each set contains a subset of a set that contains all elements. The elements can be any form of data, such as people or countries. The goal of set visualization is to visualize the relationship between the elements in the set



(a) Venn diagram (b) Euler diagram (c) matrix representation

Figure 1.2: Different visualization techniques for set systems. While Venn diagrams and Euler diagrams are based on geometric shapes, matrix representations are based on a grid. Venn Diagrams and matrix representations can contain empty regions, while Euler diagrams only contain regions for sets that have elements.

system, e.g., to highlight the common interests of users in a social network or to show trade agreements between countries. Set visualization is a challenging task because the number of sets and elements can be very large. There are various approaches and visualization techniques to tackle the given problem. While some approaches based on matrix representations are able to create visualizations of large datasets, planar drawings are often limited to a few sets. An example of different visualization techniques applied to the same set system is shown in Figure 1.2.

Venn and Euler diagrams are frequently used to visualize set systems in an intuitive way. They represent each set as a region in the plane that is bounded by a closed curve. An area in an Euler diagram that is occupied by one or more regions indicates the existence of set elements that are contained in the corresponding sets and in no other set. In contrast, if two sets have no elements in common, the corresponding regions in the Euler diagram are disjoint. In the context of Venn and Euler diagrams, the areas of the diagrams are also called *zones*. Unlike Euler diagrams, Venn diagrams can contain zones which do not contain any set elements, e.g. see purple area in Figure 1.2a. The displayed Venn diagram consists of three sets, resulting in 7 zones, of which 4 are occupied by set elements and 3 are empty. In contrast, the Euler diagram in Figure 1.2b contains only zones that are occupied by set elements. Thus, Venn diagrams can be used to highlight empty zones in set systems. This is either done explicitly with a visual encoding of the empty zones, or implicitly by placing labels for all elements in zones. However, because Venn diagrams contain every relationship between the sets, they are typically limited to a maximum of four sets.

An exemplary Euler diagram is shown in Figures 1.2b and 1.3. The Euler diagram visualizes treaties between European countries. Here, each outline represents a treaty and the countries contained by the outline are the countries that are part of the corresponding treaty. For example, the country Belgium (`BEL`) is part of all treaties except the treaty `France Monaco (1963)`. In contrast, the country Monaco (`MCO`) is only part of the treaty `France Monaco (1963)` and no other.

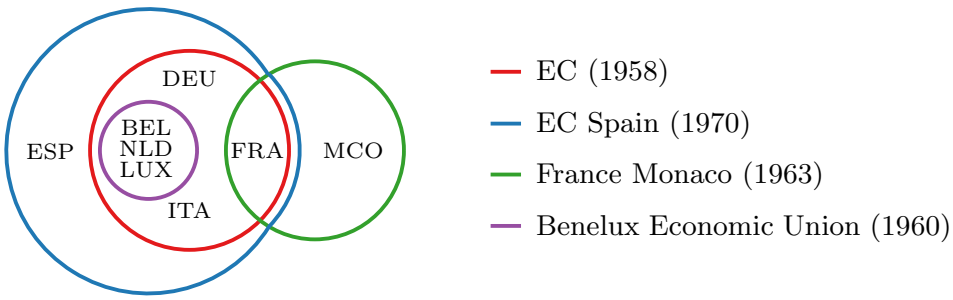


Figure 1.3: An Euler diagram visualizing treaties within Europe.

An advantage of Euler diagrams is, that they are intuitive to understand. However, they can become cluttered even for medium-sized set systems. Another problem is that for a given set system it is possible that no Euler diagram exists when the regions are required to be contiguous. In such a case, multiple approaches exist in order to draw an Euler diagram. First, the set system can be simplified. Simplification can be achieved by removing some set elements or entire sets from the set system. Second, sets can be merged to produce a drawable Euler diagram. When merging sets, most similar sets can be merged into a single set and therefore reduce the number of sets in the set system. Third, individual sets can be split into multiple regions or a single, large diagram can be divided into several smaller diagrams. These approaches are similar to map generalization, where smaller details are removed and polygons are aggregated to create a more readable map.

For a given set system there can be several topologically different drawable Euler diagrams. Moreover, every topologically different Euler diagram can be realized by an infinite number of layouts. However, the readability of the diagrams can vary considerably. In general, the outline of a set in an Euler diagram can be any shape, as long as it is free of self-intersections and contiguous. These shapes can become very complex and difficult to understand if no shape constraint is enforced. As a consequence, finding all elements of a particular set or finding elements that have common sets can become a difficult task. Therefore, it is beneficial to use simple shapes for drawing the sets in an Euler diagram to increase the readability. An established strategy is to use convex shapes such as circles, ellipses or rectangles for each set (Rodgers, 2014). However, these geometries are relatively rigid and do not allow for much flexibility when computing such an Euler diagram. As a result, fewer Euler diagrams can be drawn without violating properties of Euler diagrams. We use techniques from map generalization to simplify Euler diagrams with less strict shapes while maintaining the readability of the diagram.

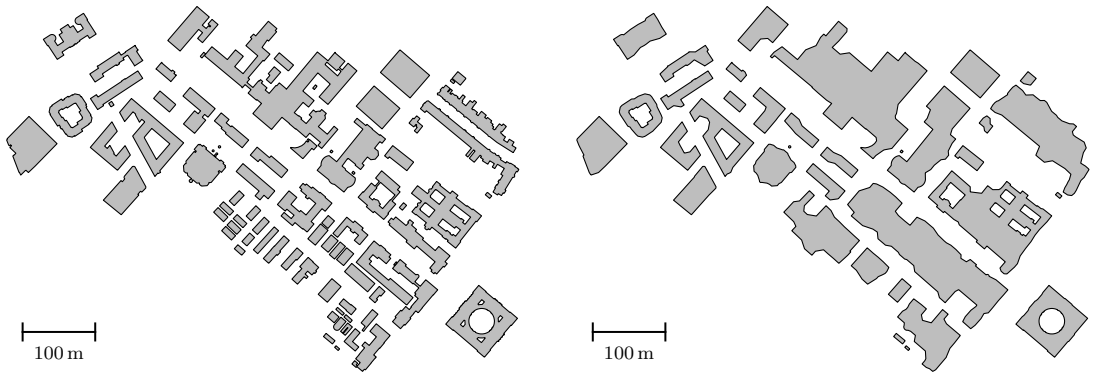
In this thesis, we address the problems of map generalization and set visualization by transforming the problems into graph-theoretic problems. We show that the proposed problems can be modeled using graphs and solved optimally by selecting subgraphs that satisfy task-specific constraints. A subgraph is a graph that contains a subset of the vertices and edges of the original graph. Despite the different applications and partially different constraints, we show that the outlined challenges in map generalization and set visualization can be optimally solved using similar techniques. For solving the graph-theoretic problem, we have to tackle two main challenges. The first major challenge is transforming the problem into a graph-theoretic problem. The vertices and edges as well as their corresponding weights need to be defined based on the particular applica-

tion. The second main challenge is to define and solve an optimization model subject to a certain objective function while incorporating different constraints that the selected subgraph must satisfy. In our algorithms, we apply two different types of constraints, among others: compactness and contiguity. We address these challenges through the use of combinatorial optimization techniques and optimization criteria from the field of spatial unit allocation. Spatial unit allocation is a well-known problem in the field of geographical information science and spatial optimization. The task is to select a subset of given spatial units, such as polygons. Typical applications of spatial unit allocation are the selection of land parcels for land use planning, forest harvest scheduling, or nature reserve allocation (McDill et al., 2002; Rahman and Szabó, 2021; Brunel et al., 2024). Most commonly across different applications, the goal is to select a subset of a set of spatial units that satisfy connectivity and/or contiguity constraints (Rahman and Szabó, 2021). In the context of this thesis, spatial unit allocation refers to the selection of a subgraph from a given graph such that the selected subgraph represents a region in an output map or visualization. When visualizing set systems, no spatial units, such as polygons, are given, but we can still use similar techniques to visualize the set systems. The vertices contained in the selected subgraph are transformed into spatial areas for the visualization. More details on the contributions of this thesis are given in the following section.

1.1 Contributions

The contributions of this thesis are multiple exact combinatorial optimization algorithms for selecting subgraphs that satisfy task-depending constraints. We formulate problems as a mathematical model and develop optimization algorithms for three different applications: map generalization, set visualization, and set system simplification. In the following, we give a brief overview of the contributions for each individual application before summarizing the key contributions of this thesis.

Our first contribution is in the field of map generalization, where we want to retrieve small scale maps from detailed data. Here, our goal is to aggregate multiple building footprints into settlement areas in order to reduce the complexity of a map for small-scale visualizations. Our algorithm leverages the concept of triangulation to partition the area between buildings into a planar subdivision. The resulting polygons, which are less detailed, are the union of a selection of triangles from the triangulation. Using a trade-off between the total area and perimeter of the selected polygons, our algorithm can compute an optimal subset of polygons with polynomial time complexity by using graph cuts. Graph cuts are well known in the field of computer vision for the task of image segmentation.



(a) input polygons (b) aggregated polygons

Figure 1.4: Aggregation of the polygons of the buildings near the street "Nussallee" in Bonn. The left image shows the input polygons, while the right image shows the aggregated polygons. The polygons are merged into several larger polygons. In addition, complex polygons are simplified, such as the polygons in the lower right and upper left.

The area-perimeter trade-off is balanced by a single parameter $\lambda \in [0, 1]$, where low λ-values lead to a small perimeter and high λ-values result in a small area of the resulting solution. This area-perimeter trade-off enables us to compute compact polygons as solutions. Hence, this single parameter controls the level of detail. We show such a polygon aggregation using an area-perimeter trade-off in Figure 1.4. In addition, we are able to report a linear-sized set of solutions that contains an optimal solution for each weighting factor λ used by our optimization function. We prove that the reported solution set has a hierarchical structure. Furthermore, we present an approximation of the solution set to report only a subset of solutions that differ by a certain threshold factor.

As a second contribution, we present a novel visualization technique for Euler diagrams using prescribed grid maps as the underlying structure. In this approach, we map every set element of a set system to a vertex of a grid graph. A requirement to our approach is that a subset of the set system forms a partitioning of the elements, which we use to form the basemap of the visualization. Cells belonging to the same set of the basemap are filled with the same color. Each remaining set of the set system which is not part of the basemap is drawn as an outline of elements in the basemap with an individual color for each set. An example of such a visualization is shown in Figure 1.5. In this example, we visualize a set system of 3 sets with 6 set elements. Since the underlying grid graph has 7 vertices, one grid cell remains empty. While our approach can generate a visualization for any given grid, our focus is on regular grids based on squares and hexagons. Similar to our approach, grid maps are a common visualization technique for spatial data that generalizes given polygons as squares or hexagons. However, our approach works on abstract data without any spatial context and creates map-like visualizations. Our approach creates a single contiguous region

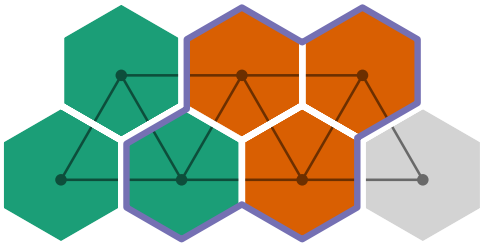


Figure 1.5: An example of a MosaicSets visualization. The set elements of the set system are mapped onto the vertices of an underlying grid graph. The grid graph is shown in black and is not displayed in the final visualization. The cells are colored according to disjoint sets which form the basemap. Additional sets are superimposed on top by drawing outlines around the set elements. Empty grid cells, e.g. bottom right cell, can be colored gray or discarded.

for each set which improves the readability. Additionally, we optimize every set to be as compact as possible. Since the basemap of our approach is a planar graph, the resulting visualizations of our algorithm are also planar. We first prove that the problem at hand is NP-hard and present a novel integer linear programming (ILP) formulation to solve the problem optimally. Additionally, we propose an optional preprocessing step to improve the computation time by grouping similar set elements during the optimization process.

In our third contribution, we present a novel approach for generating Euler diagrams for arbitrary set systems if every set has to form a single, contiguous region. To our knowledge, this is the first approach that can decide whether an Euler diagram with contiguous regions for a given set system exists and, if so, can generate the topological relations of the Euler diagram. Furthermore, our approach is able to simplify the given set system if no Euler diagram exists in order to create a drawable Euler diagram with minimal information loss. By increasing the focus on an improved layout and discarding more information, i.e. set elements and their relationships, we are able to further simplify the set system. Discarding elements results in more visually appealing and simpler Euler diagrams. Due to the close relationship between Euler diagrams and hypergraphs, we show that solving the problem of finding an Euler diagram is equivalent to finding a planar support of a hypergraph. Finding such a planar support is a well-known problem in the literature and has been shown to be NP-hard. To the best of our knowledge, there has been no proposal of an algorithm for an optimal solution of finding a planar support of a hypergraph. In addition to finding a planar support of a hypergraph, we incorporate additional constraints aimed to improving the readability of the resulting Euler diagram. These constraints are based on wellformedness criteria for Euler diagrams which have been shown to be relevant for the understanding of Euler diagrams. We focus on the criteria that were found to be most important in previous user studies. A resulting Euler diagram based on our computed planar support is shown in Figure 1.3.

Key contributions.

- Novel combinatorial optimization algorithm based on graph cuts for aggregating polygons with an area-perimeter trade-off.
- Proof of linear-size and hierarchical structure of the set of optimal polygon aggregation solutions with respect to the weighting factor of our optimization function and an optional approximation procedure of the set of solutions.
- Novel map-like visualization of Euler diagrams with contiguity constraints using grid maps of arbitrary shape.
- First approach for determining the existence of an Euler diagram for a given set system under the constraint of contiguity of each set, which is also the first formulation for finding a planar support of a hypergraph.
- Simplification of set systems with minimal loss of information under multiple constraints while maintaining drawability as an Euler diagram.

Outline. The remainder of this thesis is structured as follows. In Chapter 2, we provide an overview of related work in the fields of map generalization and set visualization. In Chapter 3, we discuss the mathematical and algorithmic background of our algorithms. In Chapter 4, we present our algorithm for aggregating polygons with an area-perimeter trade-off. In Chapter 5, we introduce our approach for visualizing set systems using prescribed grid maps. In Chapter 6, we present our algorithms for computing Euler diagrams of set systems with an optional simplification to improve readability. Finally, we conclude the thesis in Chapter 7.

Chapter 2

State of the Art in Map Generalization and Set Visualization

In this chapter, we will outline the current state of the art in the two main topics of the thesis: map generalization and set visualization. Map generalization is a well-known problem in cartography, while set visualization is a well-established field of research in information visualization. After an introduction to map generalization, we will focus on different approaches to map generalization. The main challenge in automatic map generalization is to produce a map of reduced complexity while preserving the essential information. In the field of set visualization, a similar challenge is to visualize given set-type data in a comprehensible way without losing essential information. We give an introduction to set visualization and discuss multiple approaches to this challenge.

2.1 Map Generalization

When creating maps, it is often necessary to simplify the representation of the data. Displaying every detail of the data in small-scale maps would lead to cluttered and unreadable maps. *Map generalization* is the process of simplifying the representation of geographic data while preserving the essential information. A map can be generalized by removing geographical objects, but also by grouping several objects into a single one or by simplifying given shapes. Creating readable maps is a complex task and requires the combination of different generalization operators. McMaster and Shea (1992) identified two main types of generalization operators: spatial and attributive. Spatial operators change the geometry of the objects, e.g., by simplifying, smoothing, or aggregating them (Hake et al., 2002). Attributive operators do not only take the geometry into account but also the attributes of the objects, e.g., by grouping geometric objects based on their features. In total, McMaster and Shea (1992) identified 12 different generalization

operators, which can be combined to create readable maps.

While operators like displacement or refinement can be applied to point, line, and polygon objects, other operators are specific to one or two of these types. An example for different operators for similar tasks is aggregation of points, merging of lines and amalgamation of polygons. Each of these operators has its own challenges and requires different approaches to solve them. When aggregating points, the goal is to find a single polygon or multiple polygons enclosing the input data. In contrast, merging lines is about combining multiple lines into a single one, while amalgamation of polygons is about merging multiple polygons into a single one. In this thesis, when we talk about polygon aggregation, we are referring to the amalgamation of polygons. Applying simplification operators to polygons was tackled in several works, e.g., Haunert and Wolff (2010b). For the simplification of lines, Douglas and Peucker (1973) presented an algorithm, also known as *Douglas-Peucker* algorithm, which is widely used in map generalization. The algorithm begins with a line segment between two endpoints. If any point on the line is more than a given threshold distance away from the line, the line splits at that point, and the algorithm recursively applies to the two new line segments. The result of the algorithm is a simplified line that is at most the given threshold distance away from the original line. We show the impact of four generalization operators in Table 2.1.

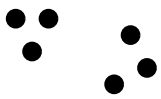
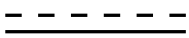
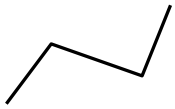
	Displacement	Simplification	Aggregation	Amalgamation
Original				
Generalized				

Table 2.1: A subset of generalization operators described by McMaster and Shea (1992). The first row shows the original objects, the second row shows the generalized objects using the corresponding operator in the header of the column.

When simplifying objects is not sufficient for a target map scale, it is often necessary to select objects and increase their size in order to be visible in smaller scale. In this process, additional space is needed in the surroundings of the object. By displacing objects, the distance between them can be increased, which increases the readability of the map. Combining several operators for increasing the quality of maps is still an open challenge in cartography. Current work by Rosenberger et al. (2025) combines selection and displacement operators to improve the readability of maps.

Automatically computing generalized maps remains a big challenge despite decades of research. The challenge lies on the one hand in the complex interplay between different processes of map generalization. A way to deal with this is to use multiagent systems for the orchestration of multiple map generalization operators (Galanda, 2003; Maudet et al., 2014). On the other hand, the challenge lies in the acquisition of cartographic knowledge in a form that can be used by a computer. Machine learning approaches have been proposed to solve this task, with a recent shift towards deep learning (Touya et al., 2019; Fu et al., 2024). Classical machine learning approaches treat input data as images and use convolutional neural networks (CNNs) to learn a generalized representation (Feng et al., 2019; Courtial et al., 2023). In contrast, modern approaches use graph convolutional networks (GCNs) (Kipf and Welling, 2017) applied to vector data (Zhou et al., 2023b; Xiao et al., 2024b). In contrast to machine learning approaches, our algorithms provide optimal solutions on vector data for our proposed problems. Polygon aggregation is relevant when generalizing categorical coverage maps (Haunert and Wolff, 2010a; Gedicke et al., 2021) or choropleth maps (Oehrlein and Haunert, 2017), where the polygons form a mosaic. These tasks are similar to districting tasks where the aim is to group small areal units to form larger regions such as electoral districts or police districts (D'Amico et al., 2002; Kim et al., 2016).

2.1.1 Aggregation and Amalgamation

In the following, we will discuss related work on the aggregation of points and polygons, as we use similar techniques in this thesis. The similarity of the two tasks is that both aim to find a single polygon or multiple polygons enclosing the input data.

Aggregation of points. A naïve method for aggregating points is to compute the *convex hull* of all input points. The convex hull is the smallest convex polygon without holes that encloses all points. However, since this may enclose large empty regions, generalizations of the convex hull such as α-shapes (Edelsbrunner et al., 1983) have been developed. As an unwanted side effect, α-shapes tend to introduce narrow bridges between two nearby point sets; see Figure 2.1. These

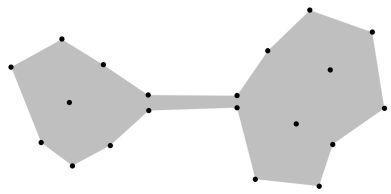


Figure 2.1: An α-shape that generates a narrow bridge between two point sets.

narrow bridges increase the complexity of the resulting polygon and can be problematic for the subsequent visualization. Such a bridge can consist of a single edge, in which case it can be easily removed (Bonerath et al., 2019), but handling bridges of multiple parallel edges is not straightforward. When the point set is triangulated with a Delaunay triangulation it is possible to compute the α-shape on the triangulation directly. This is done by selecting every triangle whose circumcircle has a radius less or equal $1/\alpha$. Consequently, we can easily compute the α-shapes for all values of α: simply add the triangles ordered by the radius of the circumcircle to an initially empty set and report the solution after each step. However, this approach has the same shortcomings as using α-shapes on the point set directly, except single edges will not occur due to the triangle-based selection process. Similar issues can arise with the concave hull introduced by Moreira and Santos (2007), which is based on k-nearest neighbors clustering. Duckham et al. (2008) defined the χ-hull (chi-hull) as a further generalization of the convex hull. First, all points are triangulated using a Delaunay triangulation. Then, all boundary edges that are longer than a threshold are removed. This procedure always returns a single polygon without holes. This may include large empty regions and does not separate groups of points from each other. To address the latter issue, Duckham et al. (2008) suggest identifying clusters in a pre-processing step. However, this does not prevent the method from covering large empty regions within a cluster.

Amalgamation of polygons. With respect to the amalgamation of polygons, Jones et al. (1995) proposed a method for merging polygons by selecting triangles of a constrained Delaunay triangulation, which they call *adopt merge amalgamation*. They do not specify the criteria for the selection of the triangles but generally recommend to use rules based on thresholds on the triangles' edge lengths. Using such rules, the adopt merge amalgamation operator has been implemented and experimentally evaluated by Li and Ai (2010). They showed that the method tends to generate narrow bridges that can consist of a single triangle touching an input polygon with only one of the triangle's vertices. The method of Li et al. (2018) overcomes this deficit by selecting sequences of triangles instead of single ones. However, the rules used to govern the selection are set up to merge polygons with parallel boundaries separated by long and narrow corridors. With this the method can be used to derive built-up areas from city blocks but not, e.g., settlement areas from footprints of detached houses. Sayidov et al. (2022) compute groups of polygons using a triangulation-based method and suggest computing a representative polygon for each group in a separate processing step. To accomplish this step, automatic typification methods can be used (McMaster and Shea, 1992; Anders and Sester, 2000; Burghardt and Cecconi, 2007). Similarly,

Steiniger et al. (2006) suggest a method that first detects groups of islands and then generates a representative polygon for each group. They define the groups with a subgraph of a minimum spanning tree and generate for each group the convex hull. Damen et al. (2008) present an approach for building generalization based on morphological operators. They combine multiple closing and opening operations to simplify but also aggregate the input polygons. To preserve the original rectangular geometry typical of buildings, they use a Minkowski sum of the input polygon and a square aligned with its boundary to implement the closing operator.

2.1.2 Continuous Generalization

Maps in the digital age are not static but can be manipulated by users in real-time to any desired map scale. As a result, multiple different levels of detail are needed for the same map. To avoid disturbance of the user experience, the map generalization should be a monotonic process, i.e., the map should continuously generalize when zooming out. Providing methods for continuous generalization is a challenging task. Such maps, which are consistent across multiple zoom levels, are called pan-scalar maps (Roth et al., 2008). Previous work on pan-scalar maps has focused, e.g., on computing active ranges for point features, where the active range of a point feature determines at which scales it is displayed (Schwartges et al., 2013). Similarly, by computing a sequence of edge removals, a step-wise generalization of a road network can be achieved (Chimani et al., 2014). Morphing techniques have been proposed to animate the transition between two representations of a line feature that differ with respect to the level of detail (Van Kreveld, 2001; Nöllenburg et al., 2008; Forsch et al., 2024). Another approach integrates multiple line generalization operations into a complex generalization with a matrix-based structure (Huang et al., 2017). For generating pan-scalar land cover maps, Peng et al. (2020) introduced an approach that computes an optimal sequence of merges of adjacent regions. Similar maps can be generated using a 3D data structure in which the zoom level is described by the z-axis (Šuba et al., 2014; van Oosterom et al., 2014; Šuba, 2017). Such pan-scalar maps can be useful for mobile and web applications with limited bandwidth (Zhao et al., 2020). First, the smaller scales can be loaded, followed by the ones with higher information density. With respect to the aggregation of individual polygons, typification can group buildings to larger polygons (Sester and Brenner, 2005). However, the resulting polygons are not completely consistent across multiple zoom levels, as smaller polygons are replaced by larger ones, which can lead to displacements. Closely related to our algorithm for map generalization is the method of Peng and Touya (2017), which iteratively adds bridges between buildings to grow settlement regions.

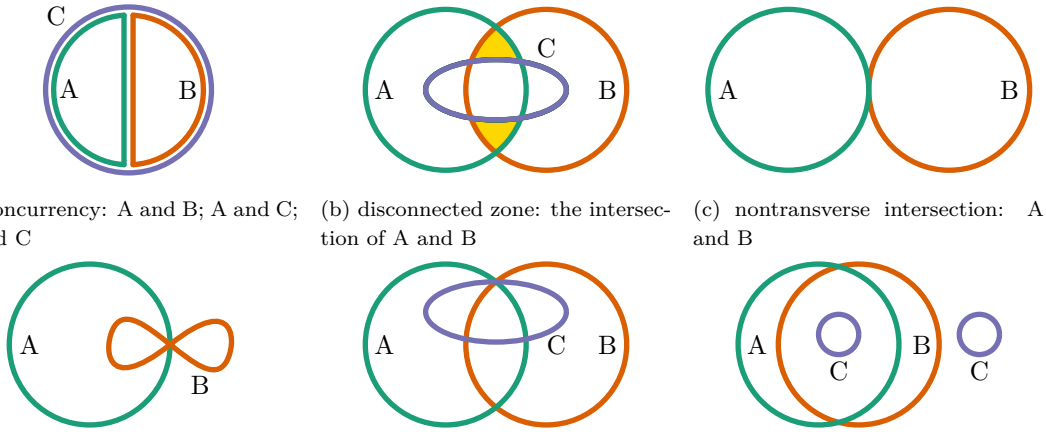
2.2 Set Visualization

Set visualization is a well-established field of research that focuses on the visualization of set systems. Set-typed data is a common data type in many domains, including biology, social networks, and computer science. By visualizing set systems, we can gain insights into the relationships between different sets and their elements. Since set systems have a high variability in their properties (e.g., number of set elements, number of relations), and the tasks a visualization has to fulfill can vary widely, a large variety of set visualization techniques exists. The survey by Alsallakh et al. (2016) introduced a classification of existing techniques. Since Euler diagrams are most related to this thesis, we will discuss them in more detail. However, we will also briefly mention other techniques that are relevant for our work. We follow the classification of set visualization techniques by Alsallakh et al. (2016).

Venn and Euler diagrams. Two standard techniques for set visualization, which were already introduced in the 18th and 19th century, are Euler and Venn diagrams (Euler, 1768; Venn, 1880). Even before the formal introduction of Venn diagrams, similar diagrams were used in the 11th century (Edwards, 2006). Both, Euler and Venn diagrams, represent each set by a closed curve and its enclosed region. In Euler diagrams, every overlap of a set of regions represents a non-empty intersection of the corresponding sets. In the context of Venn and Euler diagrams, the individual areas of the diagrams are also called *zones*. Labels for elements can be placed in the zones, however, this is not necessary to represent the set relations. In contrast, Venn diagrams show an overlap area for every possible combination of set intersections; here labels for elements are necessary to indicate which set intersections are non-empty. Euler diagrams are an intuitive way to display elements, sets, and set relations, but are mostly limited to a few sets due to clutter and drawability issues (Alsallakh et al., 2016). Yang et al. (2024) presented a tool that combines multiple set visualization techniques, including Euler and Venn diagrams, to provide a comprehensive view of different set systems.

Much research on Euler diagrams has focused on the definition of *wellformedness conditions* and how to ensure them. These conditions describe properties of Euler diagrams that target an improved comprehension of the diagrams. Typical wellformedness conditions include:

- **No concurrency**: no pair of curves run concurrently, violated in Figure 2.2a.
- **Connected zones**: each zone in the diagram is connected, violated in Figure 2.2b.
- **Transversality**: intersecting curves always intersect transversally (that is



(a) concurrency: A and B; A and C; B and C

(b) disconnected zone: the intersection of A and B

(c) nontransverse intersection: A and B

(d) non-simple curve: B.

(e) triple point: A, B and C

(f) duplicated curve label: C

Figure 2.2: Examples of wellformedness conditions following Rodgers et al. (2011).

they cross, rather than just 'touch'), violated in Figure 2.2c.

- **Simplicity**: all curves are simple curves. A curve is simple if it does not cross itself, violated in Figure 2.2d.
- **No triple points**: there are no triple points of intersection among the curves, violated in Figure 2.2e.
- **Unique curve labels**: no curve label is used more than once, violated in Figure 2.2f.

For a more formal definition of wellformedness conditions, see Stapleton et al. (2007). To verify the importance of each property, Rodgers et al. (2011) conducted a user study measuring time and errors during the completion of given tasks. Disconnected areas and concurrent curves caused significant errors and an increase in completion time. Moreover, representing a set with more than a single curve had a significantly adverse impact on task completion time.

Early approaches to Euler diagram embedding aimed to create wellformed Euler diagrams with rather strict conditions. Flower and Howse (2002) defined *concrete Euler diagrams*, which met all the wellformedness conditions above. However, strictly requiring all conditions means many set systems cannot be drawn.

Methods for generating Euler diagrams with relaxed wellformed conditions have been developed by Rodgers et al. (2008) and Simonetto and Auber (2008). However, these methods allow concurrency and duplicated curve labels. Moreover, they are heuristic and may introduce duplicated curve labels even when it is unnecessary. Since some existing methods (e.g., Simonetto et al. (2009)) yield Euler diagrams that can look distorted and stretched, EulerSmooth has been developed to smooth the curves of an existing Euler diagram (Simonetto et al., 2016). For achieving this goal, the algorithm applies a curve shortening flow approach. During this approach, the algorithm ensures that every set element stays

within the correct area using a force-directed edge-aware algorithm, ImPrEd (Simonetto et al., 2011). SPEULER constructs Euler diagrams where set elements are arranged using circular layouts (Kehlbeck et al., 2022). The resulting Euler diagrams are wellformed, which is partly a result by allowing only neighboring areas with no concurrency. RectEuler represents set-like data as Euler diagrams using rectangles as enclosing curves (Paetzold et al., 2023). The resulting Euler diagrams are wellformed. However, they split the diagram into multiple diagrams if necessary. If the set system is not to be split, rectangular Euler diagrams can fail to represent the set system without empty zones. The number of empty zones can be reduced by ordering the sets (Priss and Dürrschnabel, 2024). However, the authors note that even with an optimal ordering of sets, empty zones cannot be avoided. Additionally, there exist approaches for drawing the set elements of set systems as glyphs within the corresponding region of the Euler diagram (Brath, 2012).

Recently, Zhou et al. (2023a) presented a method for the simplification of hypergraphs and used Euler-like diagrams to visualize the results, but the simplification method does not consider the wellformedness conditions. They achieve the simplified result by merging vertices and edges of the hypergraph representation. Oliver et al. (2024) presented a method for hypergraph simplification and visualization which, however, does not use Euler diagrams. Instead, the hyperedges are represented as polygons, whose vertices are set elements of the set system. The authors first simplify the hypergraph in order to find layout for a simplified hypergraph. Then, they expand the simplified hypergraph to the original one which results in an improved layout of the original hypergraph.

Many Euler diagram techniques (Rodgers et al., 2008; Stapleton et al., 2011; Micallef and Rodgers, 2014; Kehlbeck et al., 2022) focus on showing set relations and hence do not display individual elements. There are also examples of Euler diagram techniques that explicitly visualize also the elements as small glyphs or by distributing the element names inside the corresponding regions (Simonetto et al., 2009; Riche and Dwyer, 2010; Brath, 2012; Micallef et al., 2012).

Overlay techniques. Another type of set visualization techniques are overlays, where the placement of the elements is given as input, e.g., from their spatial attributes. Overlay techniques have a limited scalability with respect to the number of elements and sets (Alsallakh et al., 2016). An example of an overlay technique showing sets as regions is MapSets (Efrat et al., 2015), which partitions the underlying map into polygonal regions, so that each region contains only elements of one set. A generalization of MapSets are ClusterSets (Geiger et al., 2021) that allow a set to be depicted by more than one polygonal region. In Bubble Sets (Collins et al., 2009) a smooth bubble-like region is computed for each set. The regions of

different sets can overlap. Alternatively, overlay techniques can indicate sets by linear spanning structures. One example is LineSets (Alper et al., 2011), which computes Bézier curves passing through the elements of each set using a traveling salesperson heuristic. A similar approach are Kelp diagrams (Dinkla et al., 2012) that visualize a set with a spanning graph structure.

Matrix-based techniques. The advantage of matrix-based techniques is that they are set visualizations that are clutter-free (Alsallakh et al., 2016). An example is ConSet (Kim et al., 2007), where the sets and elements are associated to the rows and columns of a matrix; matrix entries encode whether an element is contained in a set or not. In UpSet (Lex et al., 2014), the matrix columns represent the sets, and each row corresponds to a set intersection. The cells encode whether the corresponding set is part of the corresponding intersection. In OnSet (Sadana et al., 2014), each cell of the matrix corresponds to an element. The matrix is copied for each set such that only the cells of elements in the set are colored. Frequency grids (Micallef et al., 2012) aim at communicating set sizes. Each entry of the matrix corresponds to an element, but set systems are usually small with few overlaps. Often the elements are placed set by set in horizontal (Brown et al., 2011) or vertical order (Price et al., 2007). Also, a random order (Brase, 2009) has been considered, where the contiguity of the sets has been sacrificed. As far as we know, there exists no work on the optimization of the elements' placement for frequency grids.

Chapter 3

Algorithmic Fundamentals

As an algorithmic foundation of this thesis, we introduce the concept of graphs in Section 3.1. Graphs are essential for transforming the presented problems into graph theoretic representations. Since we utilize integer linear programming (ILP) in our algorithms related to set systems, we provide a brief introduction to integer linear programming in Section 3.2 and present the use of ILP for solving combinatorial optimization problems. Using graphs and ILP, we present spatial unit allocation and its typical use cases in Section 3.3. In Section 3.4, we conclude this chapter with an introduction to graph cuts and their applications.

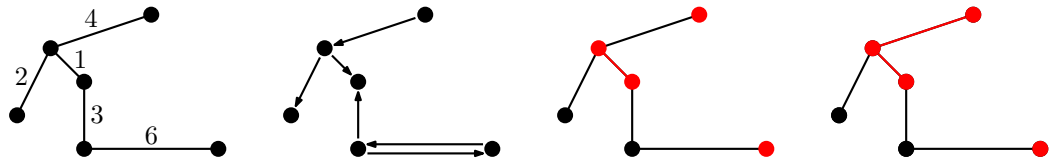
3.1 Graph Theory

In this thesis, we use graphs to model the presented algorithmic problems. Hence, we provide a brief introduction to graph theory. We both discuss basics of graphs as well their extension to hypergraphs.

3.1.1 Graph Fundamentals

Due to the broad range of graph theory, we only cover the topics that are relevant for this thesis. We refer to further literature, e.g., Cormen et al. (2022) and Harary (1969), for a more detailed introduction to graph theory.

Graphs. A *graph* $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* E. Each vertex $v \in V$ is a unique element of the graph. Each edge $e \in E$ connects two distinct vertices $u, v \in V$. Thus, each edge e can be represented by a pair of vertices (u, v). If the graph is *undirected*, the order of the vertices in the pair does not matter, i.e., we can note an edge as an unordered set of vertices $e = \{u, v\} = \{v, u\}$. In contrast, in *directed* graphs, the order of the vertices in the pair is important. Thus, the order of the vertices in the pair is



(a) weighted graph (b) directed graph (c) subgraph (d) induced subgraph

Figure 3.1: Examples of different graphs. Figure (a) shows a weighted graph, Figure (b) shows a directed graph. In (c) and (d), the subgraph and the induced subgraph are shown in red.

noted as $e = (u, v) \neq (v, u)$. The *degree* of a vertex v is the number of edges that are incident to v and is denoted by $\deg(v)$. In *weighted* graphs, each edge has an associated weight defined by a weight function $w\colon E \to \mathbb{R}$. In *unweighted* graphs, the weights of each edge is set to $w(e) = 1$. The weight can represent a cost, distance, or any other value that is associated with the edge. Similar to the vertices, the edges of a graph are unique. Between two vertices u and v there can be at most one edge in an undirected graph and at most one edge in each direction in a directed graph. If there are multiple edges between two vertices, the graph is called a *multigraph*. An undirected, weighted graph is shown in Figure 3.1a and a directed, unweighted graph in Figure 3.1b. Graphs, which are allowed to have self-loops, are called *pseudographs*.

Subgraphs. A *subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ is a graph where $V' \subseteq V$ and $E' \subseteq E$. Thus, the subgraph G' can only contain vertices and edges that are part of the graph G. It is possible to select a subgraph that contains all vertices of the original graph but only a subset of the edges. This includes the case where the subgraph does not contain any edges at all. In contrast, an *induced* subgraph $G' = (V', E')$ is a subgraph where $V' \subseteq V$ and

$$E' = \{\{u, v\} \in E \mid u, v \in V'\}. \tag{3.1}$$

Thus, the induced subgraph contains all edges of the original graph G whose endpoints are part of the vertex set V' of the induced subgraph. Examples of a subgraph and an induced subgraph are shown in Figure 3.1c and Figure 3.1d, respectively.

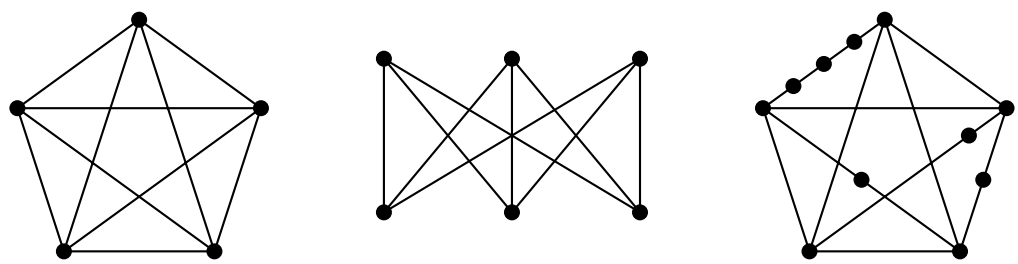
Paths and connectivity. In a graph, a *path* is defined as a sequence of k vertices $\langle v_1, \ldots, v_k \rangle$ such that each pair of consecutive vertices (v_i, v_{i+1}) is connected by an edge. A graph is *connected* if there is a path between every pair of vertices. If a graph is not connected, it consists of multiple *connected components*. Hence, two vertices u and v are in the same connected component if there is a path between them. The connected components of a graph can be computed using the breadth-first search or depth-first search algorithm within $O(|V| + |E|)$. If the

graph is unweighted and consists of a single connected component, the breadth-first search algorithm determines the shortest paths from the starting node to all other nodes in the graph. However, if the graph is weighted with non-negative edge weights, Dijkstra's algorithm can be used to compute the shortest paths from a starting node to all other nodes in the graph. In the context of cartography and navigation, the edge weights can represent distances or travel times. For more details see Cormen et al. (2022).

Complete graphs and subdivisions. A *complete* graph K_n is a graph with n vertices where every pair of vertices is connected by an edge. Since the graph does not contain self-loops, every vertex is connected to $n-1$ other vertices. Hence, a complete graph has $\frac{n\cdot(n-1)}{2}$ edges. In the context of *bipartite* graphs, which have to disjoint sets of vertices connected by edges, a complete bipartite graph $K_{m,n}$ has m vertices in one set and n vertices in the other set. Every vertex in the first set is connected to every vertex in the second set. Hence, a complete bipartite graph has $m \cdot n$ edges. The complete graphs K_5 and $K_{3,3}$ are shown in Figures 3.2a and b.

A *subdivision* of a graph $G = (V, E)$ is a graph that can be obtained from G by replacing edges with paths, i.e., introducing additional vertices of degree 2. These paths in G can be arbitrarily long and introduce additional vertices and edges into the graph. We show a subdivision of a K_5 in Figure 3.2c.

Planar graphs. A *planar* graph is a graph that can be drawn in the plane without edge crossings. The planarity of a graph can be determined by using Kuratowski's theorem. Kuratowski's theorem states that a graph is planar if and only if it neither contains a subdivision of a K_5 nor that of a $K_{3,3}$ (Kuratowski, 1930). These minimal, non-planar graphs are shown in Figure 3.2. Hence, a graph is planar if no Kuratowski subdivision is present. The planarity testing by Boyer and Myrvold (2004) solves the task of finding a Kuratowski subdivision in a graph with linear time complexity ($O(n)$).



(a) K_5 (b) $K_{3,3}$ (c) a subdivision of K_5

Figure 3.2: The non-planar graphs K_5, $K_{3,3}$ and a subdivision of a K_5. Kuratowski stated that a graph is planar if and only if it does not contain a subdivision of a K_5 or a $K_{3,3}$.

If a planar graph is drawn in the plane using random vertex positions, it is possible for edges to cross. To avoid edge crossings, it is necessary to compute a *planar embedding* of the graph in the plane. Chrobak and Payne (1995) presented an algorithm that computes such a planar embedding with linear time complexity. The resulting drawing is called a *planar drawing* and the graph with its vertex positions is called a *plane* graph.

The *faces* of the plane graph are the areas bounded by the edges of the graph. The set of faces F includes all inner faces and the outer, infinitely large face. As an example, a graph consisting of a triangle, a K_3, has three vertices and edges, and two faces, the inner and the outer face. Euler's formula states that for a connected planar graph with $|V|$ vertices, $|E|$ edges, and $|F|$ faces, the following equation holds:

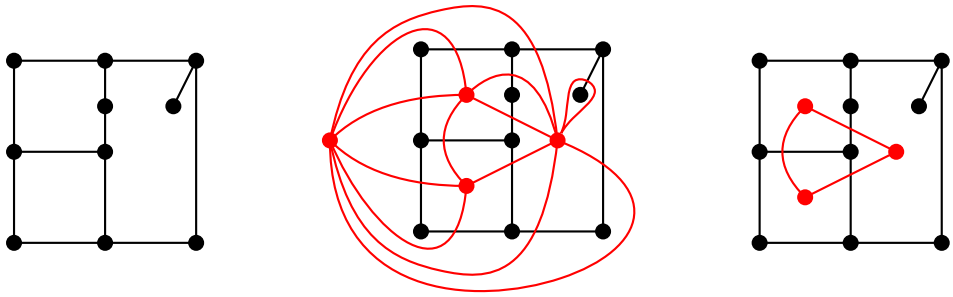
$$|V| - |E| + |F| = 2. \tag{3.2}$$

From the fact, that every face is bounded by at least three edges, i.e. a triangle, and every edge is part of at most two faces, if follows that $3 \cdot |F| \leq 2 \cdot |E|$. Using this inequality in Euler's formula, we get

$$|E| \leq 3 \cdot |V| - 6 \tag{3.3}$$

under the constraint $|V| \geq 3$. This inequality states that a planar graph with $|V|$ vertices has at most $3 \cdot |V| - 6$ edges. An advantage of this inequality is that non-planar graphs can be detected without computing a planar embedding. However, a graph can be non-planar even if this inequality holds, since this inequality only defines an upper bound for which the graph is guaranteed to be non-planar.

Dual graphs. Given a plane graph $G = (V, E)$, the representation of *dual* graphs can be used to express which faces $f \in F$ of the graph are adjacent to each other. The dual graph $G^* = (V^*, E^*)$ contains a vertex for every face f of G, including a vertex representing the outer face of G. For every edge that separates two faces in G, we add an edge to G^* connecting the two vertices representing the corresponding faces. We show an example of a plane graph and its dual graph in Figure 3.3. If both sides of an edge belong to the same face, this can be represented by a loop in the dual graph. As a result, exactly one vertex of G is in every face of G^* and vice versa. Additionally, G^* is planar as well. Hence, the number of faces of a plane graph is equal to the number of vertices of its dual graph and vice versa ($|V| = |F^*|$, $|F| = |V^*|$), while the number of edges of the plane graph is equal to the number of edges of the dual graph ($|E| = |E^*|$). When interested in the topology of the faces of a plane graph, the duplicate edges between two vertices of the dual graph and self-loops can be omitted. When removing the vertex representing the outer face from G^*, the resulting graph is called the *adjacency* graph of the plane graph.



(a) plane graph (b) dual graph (c) adjacency graph

Figure 3.3: The plane graph in black and its dual graph and adjacency graph in red, respectively. Figure (a) shows the plane graph, (b) the dual graph, and (c) the adjacency graph. The adjacency graph corresponds to the dual graph after removing the vertex representing the outer face, duplicate edges and self-loops.

Flow networks. A *flow network* is a directed graph $G = (V, E)$ where each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$ and a flow $f(u, v) \geq 0$. The flow network always contains a *source* vertex s and a *sink* vertex t. As an analogy to the flow network, we can think of a pipe system where the edges are pipes and the capacity is the amount of water that can be transported through the pipes. The flow is the amount of water that is transported through the pipes. Water is released at the source vertex and collected at the sink vertex. The goal is to send as much flow as possible from the source to the sink. To compute the flow of a network, we must ensure that the flow does not exceed the capacity of the edges:

$$0 \leq f(u, v) \leq c(u, v). \tag{3.4}$$

Additionally, flow conservation has to be ensured at each vertex $v \in V \setminus \{s, t\}$. The incoming flow at a vertex must be equal to the outgoing flow:

$$\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u). \tag{3.5}$$

The total flow of the network is the net outflow at the source or the net inflow at the sink, respectively:

$$|f| = \sum_{v \in V} f(s, v) = \sum_{v \in V} f(v, t). \tag{3.6}$$

Finding the maximum flow value $|f|$ of a flow network is a common problem in graph theory and is called the *maximum-flow* problem. The Ford-Fulkerson algorithm (Ford and Fulkerson, 1956) is an established method to solve the maximum-flow problem. The Ford-Fulkerson algorithm iteratively finds augmenting paths from the source to the sink and increases the flow along these paths. The flow is increased by the minimum capacity of the edges in the augmenting path, which

results in a time complexity dependent on the maximum flow value. More precisely, the Ford-Fulkerson algorithm has a time complexity of $O(|f| \cdot |E|)$. Orlin (2013) proposed an algorithm that is able to solve the maximum-flow problem in $O(nm)$. For planar graphs with $n = |V|$ and $m = |E|$ and $O(m) = O(n)$, they provide an algorithm that solves the maximum-flow problem in $O(n^2/\log n)$.

3.1.2 Hypergraph Drawings and Planar Supports

In the graph drawing community, a set system is usually viewed as a *hypergraph*, and different types of hypergraph drawings are considered.

Hypergraph. On a formal level, a hypergraph $\mathcal{H} = (S, C)$ consists of a vertex set S and a hyperedge set C. Each hyperedge $c \in C$ is a subset of S. Hence, a hyperedge can consist of an arbitrary number of vertices. In contrast to edges in graphs, a hyperedge can consist of a single vertex or multiple vertices. Similar to multigraphs, two vertices can be connected by multiple hyperedges. Vertices, which are contained by the same set of hyperedges are called *twins*. Consequently, a hypergraph can be used to model relationships between multiple elements. In the hypergraph representation of a set system, every vertex in S corresponds to a set element and every hyperedge in C to a set.

Support graph. A graph $G = (V, E)$ is called a *support graph* (or simply *support*) of $\mathcal{H}$ if $V = S$ and each hyperedge $c \in C$ *induces* a connected subgraph in G, i.e., the subset of edges in E that connect pairs of vertices in c is connected and spans all elements in c. Hence, the support of a hypergraph is not an induced subgraph, but a subgraph of the hypergraph where all vertices of each respective hyperedge must form a single connected component. Support graphs play an important role for visualizing hypergraphs (or set systems) since in a drawing of its support graph all hyperedges can be traced or highlighted as connected shapes, similarly to the regions representing a set in common set visualization approaches, e.g., Euler diagrams. Supports need to satisfy certain quality criteria to be considered suitable for set visualization. Most prominently, a good support graph should be planar so that the shapes created by tracing the corresponding hyperedge subgraphs intersect only if their hyperedges share common vertices.

Supports have been primarily studied from a theoretical perspective, and it is known that deciding whether a given hypergraph admits a planar support is NP-complete (Johnson and Pollak, 1987), even if the support must be 2-outerplanar (Buchin et al., 2011). In the context of Euler diagrams, it is necessary to find a *planar support* of a given hypergraph H, i.e., a support of H that can be drawn in the plane without edge crossings; see Figure 3.4. In essence, a

planar support of a given hypergraph can serve as the dual graph of the Euler diagram that is to be constructed; see Figures 3.5 and 3.6.

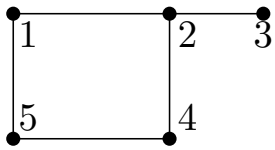


Figure 3.4: A planar support of the hypergraph with vertices $1, \ldots, 5$ and hyperedges $\{1, 2, 3\}$, $\{1, 4, 5\}$, $\{2, 4\}$, each inducing a connected subgraph.

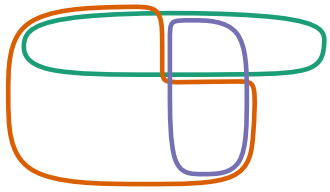


Figure 3.5: An Euler diagram whose dual graph is the planar support in Figure 3.4. The three regions correspond to the three hyperedges of the hypergraph.

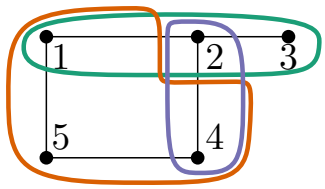


Figure 3.6: The planar support of the hypergraph in Figure 3.4 superimposed with an Euler diagram using the planar support as its dual graph.

Testing whether a hypergraph has a support that is a path, cycle, tree, or cactus (Johnson and Pollak, 1987; Korach and Stern, 2003; Brandes et al., 2011; Buchin et al., 2011) can all be done in polynomial time. Johnson and Pollak (1987) presented an efficient algorithm that yields a *tree support* if it exists, i.e., a planar support that is a tree. Following up on this study, efficient algorithms were developed that can deal with edge weights to express preferences for including certain edges in a tree support (Korach and Stern, 2003; Klemz et al., 2014). Another line of research has dealt with finding planar supports for hypergraphs with special properties. Brandes et al. (2011) presented an efficient algorithm for hypergraphs that are closed under intersections and differences, i.e., every intersection and difference of two hyperedges is also a hyperedge. Moreover, a hypergraph has a planar support if there is a non-empty intersection of all hyperedges (Chow and Ruskey, 2005), there are at most eight hyperedges (Verroust and Viaud, 2004), or the hypergraph results from the intersection of two families of regions that have a special property, called non-piercing (Raman and Ray, 2018). None of the existing methods, however, can decide for an arbitrary hypergraph whether it has a planar support.

Recently, van Bevern et al. (2024) studied the role of twins in computing planar supports. They showed that when replacing a set of twins by a single vertex, the hypergraph may cease to have a planar support. However, the proof by Johnson and Pollak (1987) shows that even for hypergraphs without twins, it is NP-hard to decide whether a planar support exists. Our method for computing a planar support of an arbitrary set system can handle hypergraphs containing twins, but we propose an optional preprocessing step to replace each set of twins with a single vertex to ensure that it is represented as a single connected zone in the output Euler diagram. In practice, this step can improve the running time of our algorithm when there are many elements with the same set membership.

Mäkinen (1990) introduced two types of drawings called *edge standard* and *subset standard*. The latter type includes *vertex-based Venn diagrams*, which were defined by Johnson and Pollak (1987) and which correspond to the type of Euler diagrams that we address with our work, i.e., Euler diagrams that are based on planar supports. Later, vertex-based Venn diagrams were generalized into *subdivision drawings* (Kaufmann et al., 2009).

A practical approach based on computing hypergraph supports and rendering the visualization in the style of a schematic metro map with sets as transit lines is MetroSets (Jacobsen et al., 2021). While the above results concern supports of abstract hypergraphs, the problem of computing supports has also been studied for spatial hypergraphs, where the vertices have fixed positions in the plane and the support graphs need to preserve these vertex positions (Castermans et al., 2019). One related result deals with the question if a given hypergraph has a tree support, where each vertex has a degree bound that must be respected by the computed tree support. Buchin et al. (2011) proved that such a support can be computed in polynomial time (if one exists).

3.2 Integer Linear Programming

Integer linear programming is a general method for solving combinatorial optimization problems (Nemhauser and Wolsey, 1988). A problem is encoded in the form of a set of variables, an objective function, and a set of constraints that have the form of an integer linear program (ILP). Following the modelling of the problem, a solver is used to compute a variable assignment that is optimal under the given constraints. The variable vector $\mathbf{x}$ of an ILP contains integer variables. The objective function and the constraints of an ILP in canonical form are $f(\mathbf{x}) = \mathbf{c}^{\mathrm{T}}\mathbf{x}$ and $A\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq 0$, respectively, where $\mathbf{c}$, A, and $\mathbf{b}$ are given as constants. An equality constraint can be encoded in this form using two inequality constraints, and a binary variable can be expressed as an integer variable with upper bound 1. Although solving an ILP requires exponential time in the worst case, there exist ILP solvers that often perform well in practice. Especially for NP-hard problems, such as finding planar embedding of hypergraphs, this approach is justifiable and often successful.

Planarity of a graph. In the field of information visualization, two ILP-based methods are most related to our set system simplification. Castermans et al. (2019) considered the problem of finding a support of a hypergraph with fixed vertex positions. The support is required to be a crossing-free straight-line graph; hence, a stricter requirement than planarity is enforced. This can easily be achieved by forbidding the selection of crossing pairs of candidate edges.

However, ensuring planarity of the support without knowing the positions of its vertices is a much more involved task.

For enforcing planarity on graphs without geometry, Chimani et al. (2019) developed an ILP for the maximum planar subgraph problem. The goal of the maximum planar subgraph problem is to find a planar subgraph which has the maximum number of edges across all possible subgraphs. In theory, it is possible to add an exponential number of constraints in order to avoid edge crossing. However, adding an exponential number of constraints is not feasible in practice and would always result in large running times. Instead, the authors use a formulation that only adds constraints if a preliminary solution contains Kuratowski subdivisions. In their work, they find all planar Kuratowski subdivision of their preliminary solution and add a constraint that for each Kuratowski subdivision at least one edge must be removed. By repeating the approach for every preliminary solution, they can ensure that the final solution is planar. We use a similar approach in our ILP formulation for the generation of Euler diagrams. We first compute a support and then check if it contains Kuratowski subdivisions. If it does, we add constraints to remove at least one edge from each Kuratowski subdivision. If it does not contain a Kuratowski subdivision, we found a planar subdivision. These constraints ensure that the final solution is planar. In contrast to the work of Chimani et al. (2019), we only add a single similar constraint at a time when simplifying the set systems. In practice, this approach is efficient and allows us to solve the problem in reasonable time.

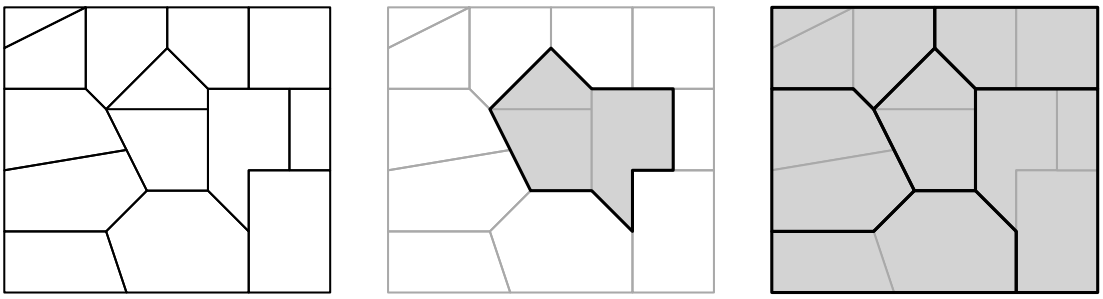
Connectivity of a graph. When modeling the embedding of hypergraphs as an ILP, the challenge lies in expressing constraints to ensure the connectivity requirement for hyperedges and the planarity requirement. Luckily, there exist constraint formulations for other graph-theoretic problems that we can adapt.

Shirabe (2005) developed a unit allocation approach that enforces the connectivity of a region resulting from the allocation of a set of faces of a planar subdivision. Although the approach is originally designed for districting problems, we can adapt it to be used for the connectivity of hyperedges in a planar support. The idea behind the formulation to model a flow network using flow variables for every edge and constraints that ensure that the flow is passed through the network towards the sink vertex. In contrast to flow networks, the network does not have a single source vertex and the sink of the flow network is not fixed. Instead, every vertex in the network, which that is not the sink vertex, is a source vertex and contributes flow to the network. The sink vertex can only accept flow and can be any of the selected vertices. In addition, a vertex is not allowed to contribute flow, if it is not part of the flow network. The flow capacity of an edge is limited by the maximum number of vertices which can be part of the flow

network. Since each vertex corresponding to an area to be part of the resulting districts adds flow to the network, which is forwarded through the network to the sink vertex, the flow network is connected. We adapt this approach to ensure the connectivity of hyperedges in our approaches. When dealing with hypergraphs, we need to implement a separate flow formulation for each hyperedge. Furthermore, the number of vertices that are part of the hyperedge is not fixed when simplifying the set system. Further details are provided in Section 5.4.1 and Section 6.4.1.

3.3 Spatial Unit Allocation

In geographical information science and spatial optimization, the term spatial unit allocation refers to a family of problems where a subset of a given set of minimal mapping units is searched Shirabe (2005). The goal of spatial unit allocation is to select a subset of minimal mapping units that form a result. The mapping units are usually defined as a set of polygons, e.g., census tracts or land use polygons. An example of such mapping units is shown in Figure 3.7a. The result can be a single, contiguous region or multiple regions. In Figure 3.7b, three mapping units are selected to form a single polygon as a result. Typically, the selection of mapping units is based on an optimization minimizing or maximizing an objective function. Depending on the application of the approach, the objective function can be a single criterion or a combination of multiple criteria. In addition to the objective function, it is possible to introduce additional constraints. These constraints depend on the use case of the approach and the requirements for the resulting geometry or additional attributes.



(a) minimal mapping units (b) spatial unit allocation (c) districting

Figure 3.7: Spatial unit allocation. Figure (a) shows a set of minimal mapping units, e.g., census tracts or land use polygons. In (b), three minimal mapping units are selected via spatial unit allocation to form a result. Figure (c) shows the districting of mapping units into multiple contiguous regions with similar area. Here, every mapping unit is assigned to exactly one region.

Compactness. An important selection criterion in spatial unit allocation is the computation of compact regions. Rahman and Szabó (2021) present a review of several spatial unit allocation approaches and find that compactness is the most commonly used criterion across applications. The compactness can be expressed with various measures, many of which are based on the area A and the perimeter P of a resulting region. For example, the Polsby-Popper score of a region (Polsby and Popper, 1991) is defined as

$$PP = 4\pi \frac{A}{P^2} \tag{3.7}$$

and the Schwartzberg score (Schwartzberg, 1965) as the square root of the Polsby-Popper score. Note that both scores evaluate to one for disks and attain values close to zero for highly non-compact shapes. Furthermore, if the area A is fixed, maximizing a shape's Polsby-Popper or Schwartzberg score reduces to minimizing its perimeter. Another general approach to measuring a region's compactness is based on eccentricities, i.e., distances between a center of a result and the areas assigned to it (Hess and Samuels, 1971). For results that consist of several regions, the compactness can be expressed as the sum of the compactness of the individual regions. Alternatively, we can achieve compact solutions by minimizing the number ob neighboring spatial units (Aerts and Heuvelink, 2002). This approach is similar to minimizing the perimeter of a region in cases where all borders of mapping units have the same length.

Spatial unit allocation approaches. For solving the optimization problem, integer programming is an established technique. Aerts et al. (2003) present an approach for multi-site land use allocation using integer linear programming. They present constraints for contiguity and compactness of the results while minimizing displacement on a regular grid. However, the contiguity constraints require the use of such a regular grid and are not applicable to arbitrary geometry. Other approaches in the context of habitat allocation and biodiversity conservation use quadratic integer programming in their objective function of constraints to model the contiguity of the output regions (Nalle et al., 2003; Billionnet, 2013). This approach allows the authors to prioritize contiguity without strictly enforcing it.

Unlike integer linear programming, where each variable can take only integer values, mixed integer linear programming (MILP) allows for continuous variables. McDill et al. (2002) present an approach to harvest scheduling using MILP. They present two formulations for selecting adjacent land units whose total area does not exceed a predefined threshold. To model the contiguity of regions, they use a path algorithm that selects up to four management units. The selection is made by selecting adjacent units until the total area exceeds the threshold, at which

point a constraint is added. Brunel et al. (2024) present an approach to allocate contiguous reserves using MILP which are gap free, i.e. the resulting reserves do not contain holes. Furthermore, they produce compact reserves by minimizing the perimeter of the reserves. Producing compact regions is also a goal of our approaches explained in Chapters 4 and 5. Brunel et al. (2024) state their method is not applicable to large instances (> 500 units), while our approach to polygon aggregation which is explained in Chapter 4 can be computed in subquadratic time complexity and scales for larger instances ($> 100,000$ units).

In contrast to using integer programming to solve the spatial unit allocation optimization problem, it has been shown that the use of genetic algorithms can be advantageous (Li and Parrott, 2016; Gao et al., 2021). Using genetic algorithms, the authors can solve multi-objective optimization problems and find a set of regions that are optimal with respect to multiple criteria. The strength of genetic algorithms is their dramatically improved scalability compared to most ILP-based methods. However, the use of genetic algorithms is not suitable if every constraint has to be strictly modeled and must not be violated. Additionally, genetic algorithms are not guaranteed to find the optimal solution. Xiao et al. (2024a) propose a different approach by starting with multiple random initializations of the same allocation problem. By swapping regions within solutions and combining results from different initializations, they claim to find near-optimal solutions.

3.3.1 Districting

The delineation of districts is a common problem in geographical information science and planning that is usually referred to as districting or zoning; for a detailed discussion we refer to Ríos-Mercado (2020). Applications of districting include but are not limited to the definition of electoral districts (Validi et al., 2021; Zhang et al., 2024), school districts (Caro et al., 2004), and ticket zones for public transportation systems (Tavares-Pereira et al., 2007). Typically, the goal is to form districts automatically based on a given partition of the plane. For example, census tracts are grouped to electoral districts. Therefore, it is a subtask of spatial unit allocation, the goal of which is to select a minimal set of mapping units to produce a result. In addition to the objective of spatial unit allocation, e.g. producing compact solutions, the resulting districts are required to be contiguous. For their typical application of partitioning population in districts, e.g. creating electoral districts, they should satisfy a number of additional constraints, such as population balance or the preservation of existing boundaries (Tong and Murray, 2012).

Contiguity. A district is *contiguous* if every two points in it are connected via a curve that is entirely within that district. Thus, each district is a connected component, and contiguity is a qualitative property of a district. In this context, we want to highlight the difference between connectivity and contiguity that we use in this thesis. We use *contiguity* to refer to the property of a region which can be represented by a single polygon. When using *connectivity*, we refer to the property of nodes in a graph which are connected by a path. In contrast to contiguity, compactness is a quantitative property.

Districting in cartography. In the context of information visualization and cartography, districting methods have been proposed for map generalization tasks. Oehrlein and Haunert (2017) presented a method for grouping the areas of a choropleth map to larger areas in order to obtain a less fine-grained choropleth map. Similar approaches have been used for land-use and land-cover maps (Haunert and Wolff, 2010a; Gedicke et al., 2021). Although the constraints and objectives of districting depend on the concrete task at hand, some approaches have turned out to be rather generally applicable. This holds for meta-heuristics such as simulated annealing (Ricca and Simeone, 2008) or evolutionary algorithms (Tavares-Pereira et al., 2007). These, however, usually require a feasible start solution, which is not easy to obtain for Euler diagrams. Therefore, we choose an approach based on integer linear programming over meta-heuristics, which has the additional advantage that it guarantees optimal solutions.

Impact on this thesis. For drawing contiguous diagrams in our set visualization problems, we adapt a model developed for districting by Shirabe (2005, 2009) that is based on flow networks. Additionally, we transfer the idea of producing compact districts to the research field of polygon aggregation and set visualization. In contrast to spatial unit allocation problems in cartography, tasks in information visualization do not provide any geometric objects which can be used as minimal mapping objects. However, in our work, the vertices of our graph are becoming the areas of the visualization. Thus, by selecting a subset of vertices, we can also select a subset of areas. We provide formulations which can transfer proven methods for ensuring contiguity of the individual sets to our problems.

3.4 Graph Cuts

The idea behind graph cuts is to partition a graph into two disjoint graphs by removing a set of edges. If the graph is unweighted, the goal is to find a cut that minimizes the number of edges between the two partitions. If the graph is weighted, the goal is to find a cut that minimizes the sum of the weights of the

edges between the two partitions. First, we introduce the concept of graph cuts and then discuss their application in image segmentation. Afterwards, we present the use of graph cuts in additional fields of research, such as climate models or 3D point cloud reconstruction.

Graph cuts. Let $G = (V, E)$ be an undirected, weighted graph with the vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{\{v_i, v_j\} \mid v_i \in V, v_j \in V\}$ with an edge weight $w\colon E \to \mathbb{R}_{\geq 0}$. A graph cut removes a set of edges $C \subseteq E$ such that the graph is partitioned into two disjoint subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and $E_1 \cup E_2 \cup C = E$. The cut capacity is defined as the sum of the weights of the edges in C.

$$w(C) = \sum_{e \in C} w(e) \tag{3.8}$$

When computing a minimum cut, we minimize the cut capacity (Peng et al., 2013).

Image segmentation. Graph cuts are widely used for image segmentation (Wu and Leahy, 1993; Shi and Malik, 2000), stereo matching (Bleyer and Gelautz, 2007; Barath and Matas, 2018) or medical image segmentation (Chen and Pan, 2018). A survey of Peng et al. (2013) provides an overview of graph-theoretic approaches to image segmentation. The survey is not limited to binary segmentation, but also covers multi-label segmentation and the use of other graph-based methods such as tree-based methods. The authors showed that graph cuts are well suited for this task and are able to solve the problem efficiently.

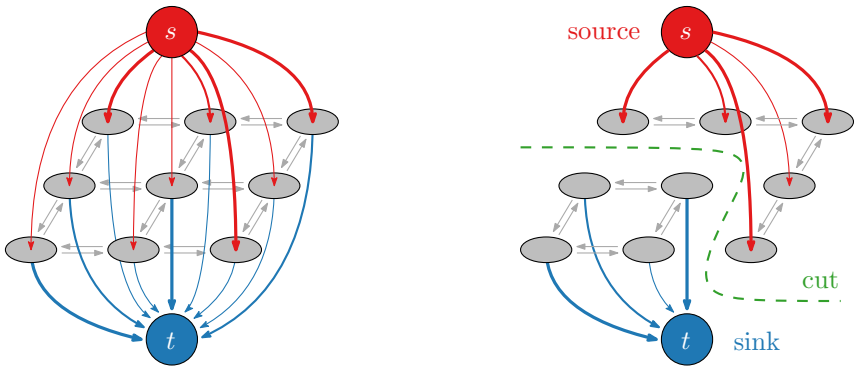
To apply graph cuts to images, we first define the vertices and edges of our graph G. First, we introduce a vertex in the graph for every pixel in the image. Additionally, we need to define which subgraph will represent the foreground and the background after cutting the graph. To accomplish this, we add two artificial vertices s and t, called the *source* and *sink* vertices, respectively.

$$V' = V \cup \{s, t\} \tag{3.9}$$

The basic set of edges connects every vertex of the graph representing a pixel with the source and sink.

$$E' = \{\{s, v\} \mid v \in V\} \cup \{\{v, t\} \mid v \in V\} \tag{3.10}$$

Computing an s-t-cut, we separate the source vertex s from the sink vertex t. We call the connected component of the graph containing the source vertex the *source component* and the component containing the sink vertex the *sink component*. The resulting subgraphs have at least one vertex, i.e. the source or sink, and can



(a) graph G (b) a cut on G

Figure 3.8: A graph cut based on the model by Greig et al. (1989). Edge weights are represented by thickness. After cutting the graph, the source and sink components are separated.

have no edges. Due to the theorem of Ford and Fulkerson (1956), the minimum cut problem is equivalent to the maximum-flow problem. In the maximum-flow problem, the goal is to maximize the flow between two vertices in a graph. The edges that restrict the flow are the same edges as in the minimum cut problem. Orlin (2013) presented an algorithm that solves the maximum-flow problem in $O(nm)$ time, where n is the number of vertices and m is the number of edges. For planar graphs, the property $m = O(n)$ holds resulting in the time complexity $O(n^2/\log n)$.

Since the presented basic formulation does not incorporate any information of neighboring pixels, Boykov and Veksler (2006) introduced a method to incorporate information of adjacent pixels. The additional information is encoded in edges between vertices representing adjacent pixels in the image; see Figure 3.8. The weights of the edges depend on the similarity of the adjacent pixels and are based on a maximum a posteriori approach, which was introduced by Greig et al. (1989). The approach of Greig et al. (1989) was generalized to using energy minimization for adjacent pixels by Kolmogorov and Zabin (2004). They introduced a novel term for the energy minimization utilizing triplets of pixels. With respect to the automatic tuning of a parameter that balances two objectives, the work of Peng and Veksler (2008) is most related to ours. While they focus on the development of quality metrics for the evaluation of image segmentation solutions, they simply use a constant step width to sample different values for the parameter of a weighted-sum model.

Another field of research is the use of graph cuts for segmenting images under the constraint of connected components. Vicente et al. (2008) proposed a method that leverages Dijkstra's shortest path algorithm to enforce connectivity constraints. They require that every pair of pixels in the segmentation set has a path in the graph that connects them. However, they prove that the problem is NP-hard and propose a heuristic to solve the problem.

Other raster data. While image segmentation is the most common application of graph cuts, other applications have been proposed for the use with raster data. Clark et al. (2012) showed an approach based on graph cuts to merge two images into one. They use an energy based cost function which takes into account the color and gradients of the images. The authors show that the choice of the cost function greatly influences the result. An energy minimization cost function was also used by Thao et al. (2022) to combine two climate models using graph cuts. They fit the models to a reference model for training and use a data term and a smoothness term in the energy function.

Vector data and point clouds. In addition to their broad application to raster data, graph cuts have been applied to vector data and point clouds in the past. While images are clearly structured as a grid, vector data and point clouds do not have a regular structure. Abrahamsen et al. (2020) used graph cuts on vector data to separate sets of polygons into interior disjoint polygons. They minimized the length of the separating boundary between the two sets of polygons. Unlike the method we developed for bicriteria shapes, they did not consider the area and the resulting compactness of the polygons. Sedlacek and Zara (2009) applied graph cuts on 3D point clouds to reconstruct 3D models. In contrast to energy models of image segmentation, they use a cost function based on the Euclidean distance between points in the point cloud. Another approach dealing with vector data is presented by Zebedin et al. (2008). Their goal is to create a 3D model of a building from aerial laser scanning data. In a first processing step, they extract the outline of the building and the roof structure. In a second step, they fuse the boundary and roof information using graph cuts with energy minimization. The authors show that the graph cut approach is able to create a more accurate 3D model than other methods. Another segmentation approach using graph cuts is presented by Nunes et al. (2022). They apply graph cuts to features generated by a convolutional neural network to segment instances in images. As a result, they do not need to provide labels to train the network. The authors note that this is particularly useful for training scenarios where not every instance class is correctly labeled in the neural network training data.

Chapter 4

Map Generalization Using Graph Cuts

This chapter is based on the joint work with Anne Driemel, Herman Haverkort, Heiko Röglin and Jan-Henrik Haunert (Rottmann et al., 2025) which is an extended version of a previous work of the same authors (Rottmann et al., 2021). The idea behind bicriteria shapes was born in a discussion among all authors. While Herman Haverkort improved the proofs in this chapter, I (Peter Rottmann) implemented the algorithm, conducted the experiments, provided the figures and was supervised by Jan-Henrik Haunert while writing.

An important task in pattern recognition and map generalization is to partition a set of disjoint polygons into groups and to aggregate the polygons within each group into a representative output polygon. We introduce a new method for this task, called bicriteria shapes: following a classical approach, we define the output polygons by merging the input polygons with a set of triangles that we select from a conforming Delaunay triangulation of the exterior of the input polygons. The innovation is that we control the selection of triangles with a bicriteria optimization model that is efficiently solved via graph cuts; in particular, we minimize a weighted sum that combines the total area of the output polygons and their total perimeter. In a basic problem, we ask for a single solution that is optimal for a given parameter value which controls the level of generalization. In a second problem, we ask for a set containing an optimal solution for each possible value of the parameter. We discuss how this set can be approximated with few solutions and show that it is hierarchically nested. An evaluation with building footprints as input and a comparison with α-shapes based on the same underlying triangulation conclude the chapter. An advantage of bicriteria shapes compared to α-shapes is that the sequence of solutions for decreasing values of the parameter is monotone with respect to the total perimeter of the output polygons, resulting in monotonically decreasing visual complexity.

4.1 Introduction

Map generalization is the process of deriving a less detailed map from a given map. It consists of several subtasks such as object selection, simplification, aggregation, and displacement. In this chapter, we address the task of detecting polygon groups (i.e., spatial clusters of polygons) and aggregating each group into a single polygon, which we will simply refer to as polygon aggregation. This task is relevant, for example, to derive settlement areas from mutually disjoint building footprints.

A popular method for aggregating polygons is the *adopt merge amalgamation operator* proposed by Jones et al. (1995), which is based on a conforming Delaunay triangulation of the space not covered by the input polygons; see Figure 4.1. The approach is to select a set $T' \subseteq T$ from the set T of triangles of the triangulation to glue groups of input polygons together. More precisely, the contiguous regions in the union of the triangles in T' and the input polygons constitute the output polygons. While Jones et al. (1995) left it largely open how the selection T' is computed, we present a new method that computes T' by optimization.

Design decisions. Our work aims to overcome the problem raised by Li et al. (2018), who mentioned that the triangulation-based aggregation of polygons has often been discussed at a general conceptual level and that difficulties arise in the detailed specification. We implement the general approach based on the two overarching criteria of map generalization (Burghardt et al., 2007): While on the one hand, map generalization aims to preserve the information given with the input map, on the other hand, it aims to ensure the legibility of the output map. In our method, the preservation of information is considered by minimizing the total area of the output polygons, meaning that only little area should be added to the input polygons when merging the selected triangles with them. Legibility is considered by minimizing the total boundary length (or *perimeter*) of the output polygons, which can be considered as an implementation of Tufte's minimum-ink principle (Tufte, 1992). A parameter λ, which we call *balance factor*, is used to combine these objectives with a weighted sum. To formalize this, we refer to $A(S)$ as the area and $P(S)$ as the perimeter of the union of all polygons in a set S, where the union can be a polygon or a multipolygon. For a single polygon p we simply use $A(p)$ and $P(p)$ to refer to its area and perimeter, respectively. With this we state the first problem that we aim to solve as follows.

Problem 1. *Given a set B of mutually disjoint simple polygons, a set $S \supseteq B$ of n simple polygons that constitute a planar subdivision, and a balance factor $\lambda \in [0, 1]$, select a set S' with $B \subseteq S' \subseteq S$ minimizing $f_\lambda(S') = \lambda \cdot A(S') + (1-\lambda) \cdot P(S')$.*

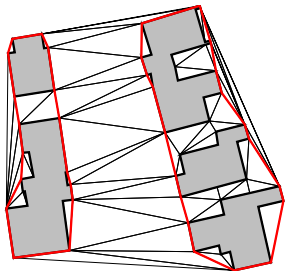


Figure 4.1: Input polygons (filled gray) aggregated to larger ones (red lines).

In our application, B is the set of input polygons, e.g., building footprints or islands. The set S contains all polygons of a planar partition, including the polygons in B and the triangles of a triangulation partitioning the space not covered by the input polygons. Note that the restriction to a triangulation is not necessary, i.e., one may use any other partition of the plane instead. The requirement $B \subseteq S' \subseteq S$ means that the input polygons in B have to be in the selection S'. The balance factor λ combines the two objectives with a weighted sum, yielding the overall objective function f_λ. To state that a solution to Problem 1 is optimal for a certain balance factor λ, we refer to it as a λ-*optimal solution*.

Any algorithm solving Problem 1 can be used to compute a clustering that is optimal for a prescribed value of the parameter λ. Such an algorithm could be used to derive a single output representation of the input data. However, setting the free parameter of the model may be challenging. Moreover, a single clustering does not provide representations for multiple scales. Therefore, we introduce a second problem to obtain a *parameter-free hierarchical clustering method*.

Problem 2. *Given a set B of mutually disjoint simple polygons and a set $S \supseteq B$ of n simple polygons that constitute a planar subdivision, find a set containing for every $\lambda \in [0, 1]$ an optimal solution to Problem 1.*

Obviously, by not requiring a pre-set value for λ, our method becomes parameter free. However, it is not as obvious that the solutions in the result set of Problem 2 constitute a hierarchical clustering. We refer to Section 4.2.2 for a detailed proof of the hierarchically nested structure of the solutions.

The hierarchically nested structure of solutions will admit a gradual transformation of the data from fine to coarse or vice versa, which is a prerequisite for zoomable maps (Sester et al., 1998). Very commonly, the hierarchy of different aggregation levels is computed once and stored in a data structure, such that a map at a user-queried scale can be retrieved at an interactive performance (Timpf and Frank, 1995). Moreover, after computing a set of solutions containing an optimal solution for each value of the parameter λ, we can infer a suitable value from a reference solution of a fixed scale: Simply find the λ-optimal solution that is most similar to the reference solution (with respect to an appropriate similarity

measure) and select the corresponding value for λ. The inferred parameter value can then be used to solve also other problem instances. Instead of relying on a reference solution, one could also employ a human expert (i.e., a cartographer) to let him or her select the best solution in the result set. In this scenario, however, it is crucial that the result set is not too large as the human expert can inspect only a limited number of solutions. Therefore, we also address the problem of approximating the result set of Problem 2 with few solutions.

Contributions and outline.

1. In Section 4.2, we present algorithms for solving the problems defined above and discuss important structural properties of the output polygons, which in the following we refer to as *bicriteria shapes*. In particular, we prove the hierarchical structure of the solutions for different parameter values.

2. In Section 4.3, we show through a detailed experimental evaluation that our method can delineate multiple settlements from each other by clustering a given set of buildings into multiple groups. Moreover, we compare our method with α-shapes (Edelsbrunner et al., 1983). Finally, we show how to preserve the characteristic edge directions of the input polygons by using a planar partition that is not a triangulation.

Note that with bicriteria shapes, we aim to introduce a method that can be applied to polygon data in general, regardless of the types of objects represented by the input polygons. We use the grouping and aggregation of building footprints as a case study but do not aim to model criteria that are relevant specifically for buildings. Our method is thus comparable with other basic geometric algorithms, such as α-shapes, which is a widely used approach for generalization. However, there is no simple relationship between the complexity of α-shapes and the parameter α. More precisely, when decreasing α, the input polygons are hierarchically aggregated into larger polygons, but the total perimeter of the output polygons does not decrease monotonically. Hence, in contrast to bicriteria shapes, α-shapes do not solve the two problems that we defined. We show the differences between α-shapes and bicriteria shapes in a comparative evaluation. Finally, we conclude this chapter in Section 4.4 and give recommendations for future research.

4.2 Methodology

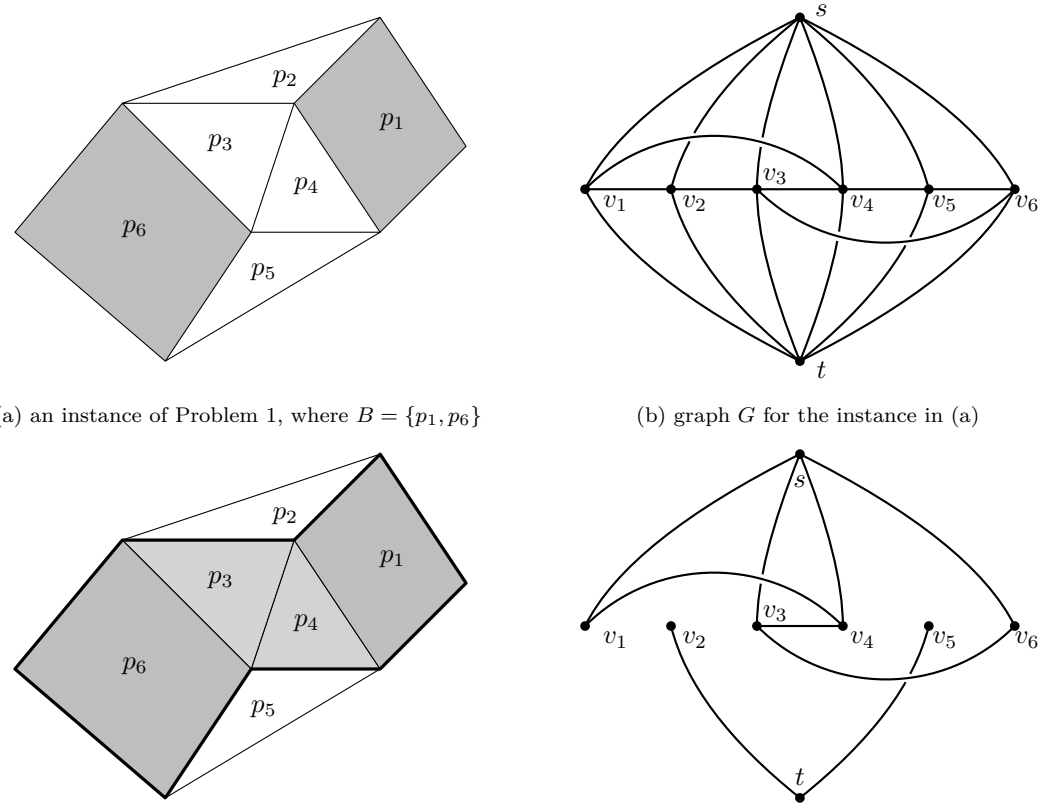
In this section, we present our approach using graph cuts. First, in Section 4.2.1, we deal with Problem 1, which asks for a selection of polygons minimizing our

bicriteria cost function. Then, in Section 4.2.2, we address Problem 2, which asks for a λ-optimal solution for every λ in $[0, 1]$. We show that a linear-size solution set exists whose solutions form a hierarchical structure. Moreover, we state how a suitable approximation of the solution set can be computed.

4.2.1 Graph Cut

For solving Problem 1 with graph cuts, we set up an undirected weighted graph $G = (V, E)$ modeling all feasible solutions as well as our minimization goal. This approach is illustrated in Figure 4.2 and is described in detail in the following.

As starting point we use the adjacency graph $G' = (V', E')$ of the planar subdivision given with the set S of polygons. Assuming that the polygons are numbered in an arbitrary order as $p_1, \ldots, p_n$, we refer to the corresponding nodes in V' as $v_1, \ldots, v_n$. The edge set E' contains an edge $\{v_i, v_j\}$ for every two polygons p_i and p_j whose boundaries share at least one line segment. We define the node set of G as $V = V' \cup \{s, t\}$, where s is a node called *source* and t a node called *sink*. The edge set E of G contains all edges in E' as well as, for $i = 1, \ldots, n$, the two edges $\{s, v_i\}$ and $\{v_i, t\}$; see Figures 4.2a and b. An s-t-cut in G is a



(a) an instance of Problem 1, where $B = \{p_1, p_6\}$

(b) graph G for the instance in (a)

(c) instance after min cut, where $S' = \{p_1, p_3, p_4, p_6\}$

(d) graph G after cut for the instance in (c)

Figure 4.2: Algorithmic solution of Problem 1 via a graph cut.

set of edges whose removal from G causes s and t to be in different connected components. We solve Problem 1 by defining an edge weighting $w\colon E \to \mathbb{R}_{\geq 0}$ and computing a *minimum* s-t-cut in G, i.e., an s-t-cut in G of minimum total edge weight.

Formally, for any s-t-cut $C \subseteq E$, we define its weight as $w(C) = \sum_{e \in C} w(e)$ and the graph $G_C = (V, E \setminus C)$. We call the connected component of G_C containing s the *source component* and the connected component of G_C containing t the *sink component* of C. Moreover, we refer to the set of polygons represented by nodes in the source component as the solution S' *modeled* by C; see Figures 4.2c and d. It remains to ensure that any solution modeled by a *minimum* s-t-cut in G is feasible and optimal with respect to Problem 1. For this we define the edge weighting w as follows:

- For every edge $e = \{v_i, v_j\}$, we set $w(e) = (1 - \lambda) \cdot \ell(p_i, p_j)$, where $\ell(p_i, p_j)$ is the length of the common boundary of polygons p_i and p_j.

- For every node v_i with $p_i \in B$, we set $w(\{s, v_i\}) = \infty$ and $w(\{v_i, t\}) = 0$. This avoids that $\{s, v_i\}$ is selected for the cut and thus ensures that v_i is in the source component. (In practice, we use a floating-point number format with a special value representing ∞.)

- For every node v_i with $p_i \notin B$, we set $w(\{s, v_i\}) = 0$ and $w(\{v_i, t\}) = \lambda \cdot A(p_i) + (1 - \lambda) \cdot \ell(p_i)$, where $\ell(p_i)$ is the length of the boundary between p_i and the outer face (0 if p_i and the outer face are not adjacent).

For computing a minimum s-t-cut and the corresponding optimal solution to Problem 1 we then use a standard graph cut algorithm. The corresponding theorem states that the approach based on a graph cut indeed solves Problem 1:

Theorem 1. *The solution modeled by any minimum s-t-cut in G is an optimal solution to Problem 1. This allows Problem 1 to be solved in $O(n^2/\log n)$ time.*

Proof. We prove that (i) each selection S' with $B \subseteq S' \subseteq S$ is modeled by an s-t-cut in G whose total weight is $\lambda \cdot A(S') + (1 - \lambda) \cdot P(S') = f_\lambda(S')$ and (ii) each s-t-cut in G of total weight $W \neq \infty$ models a solution whose objective value measured with f_λ is at most W. This together implies that any solution modeled by a minimum s-t-cut in G is an optimal solution to Problem 1.

To show (i), let S' be an arbitrary solution with $B \subseteq S' \subseteq S$ and C the cut defined as follows. For each $p_i \in S'$, we add edge $\{v_i, t\}$ to C, which amounts to weight $\lambda \cdot \sum_{p_i \in S'} A(p_i) + (1 - \lambda) \cdot \sum_{p_i \in S'} \ell(p_i)$. Moreover, we add each edge $\{v_i, v_j\} \in E'$ with $p_i \in S'$ and $p_j \notin S'$ to C, which amounts to weight $(1 - \lambda) \cdot \sum_{\{v_i, v_j\} \in E''} \ell(p_i, p_j)$ where $E'' = \{\{v_i, v_j\} \in E' \mid p_i \in S' \wedge p_j \notin S'\}$. The set C

is an s-t-cut in G modeling S' because the nodes for polygons in S' plus node s constitute the source component of C. The weight of C is

$$\begin{aligned} w(C) &= \lambda \cdot \textstyle\sum_{p_i \in S'} A(p_i) + (1-\lambda) \cdot \sum_{p_i \in S'} \ell(p_i) + (1-\lambda) \cdot \sum_{\{v_i,v_j\} \in E''} \ell(p_i, p_j) \\ &= \lambda \cdot A(S') + (1-\lambda) \cdot \Big(\textstyle\sum_{p_i \in S'} \ell(p_i) + \sum_{\{v_i,v_j\} \in E''} \ell(p_i, p_j) \Big) \\ &= \lambda \cdot A(S') + (1-\lambda) \cdot P(S') = f_\lambda(S'). \end{aligned} \tag{4.1}$$

To show (ii), we consider an arbitrary s-t-cut C' in G of total weight $w(C') = W \neq \infty$. Let S' be the solution modeled by C'. Because of $w(C') \neq \infty$, S' satisfies $B \subseteq S' \subseteq S$. Now, let C be the cut for S' as defined in the proof of (i). As argued before, C models S' and its weight equals the objective value of S', i.e., $w(C) = \lambda \cdot A(S') + (1-\lambda) \cdot P(S')$. Moreover, the weight of C is at most the weight W of C', since every edge in C is included in C' as well, which can be seen as follows. Assume that there exists an edge $e = \{u, v\}$ with $e \in C$ and $e \notin C'$. Because of $e \in C$ and the way we constructed C, one of the nodes u and v lies in the source component of C and the other one in the sink component of C. Because of $e \notin C'$, u and v lie in the same connected component of $G_{C'}$. This contradicts the assumption that C and C' model the same solution. Therefore, the assumption $e \in C$ and $e \notin C'$ must have been false. This allows us to conclude that $w(C') \geq w(C) = \lambda \cdot A(S') + (1-\lambda) \cdot P(S')$.

Currently, the fastest algorithm for computing a minimum s-t-cut in a graph with $O(n)$ edges runs in $O(n^2/\log n)$ time (Orlin, 2013). This result applies to our case since G' is planar (which implies that it has $O(n)$ edges) and since G has only two more edges for each node of G'. The running time for computing the cut dominates the running times for computing G from the input and generating the output selection from the cut (e.g., via a depth-first search in the graph without the cut edges, starting from the source). Hence, the overall running time for solving Problem 1 is $O(n^2/\log n)$. □

4.2.2 Hierarchical Structure of Solutions for Multiple Parameter Values

Before going into the details of the computation, we recall that, by definition of Problem 2, if for a fixed λ there are multiple optimal solutions, only one of them has to be included in the result set. This corresponds to finding what is called the set of all *extreme nondominated points*, which together with the *nonextreme nondominated points* constitute the *Pareto-frontier* (Bökler and Mutzel, 2015); see Figure 4.3. The focus on *extreme nondominated points* admits an efficient computation via graph cuts and results in a hierarchically nested structure. This result is summarized with the following theorem, which we first prove and then discuss for a small example.

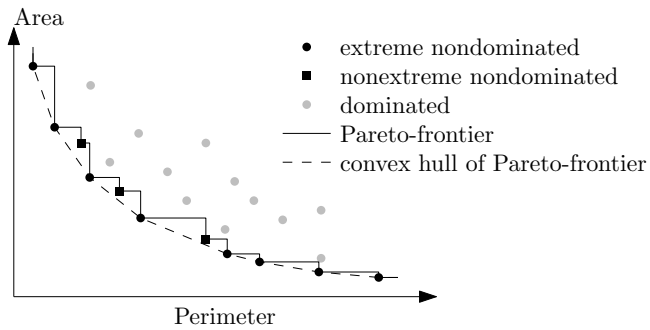


Figure 4.3: Different types of solutions and Pareto-frontier of a bicriteria optimization problem.

Theorem 2. *For an instance of Problem 1, let S_1 be a λ_1-optimal solution and S_2 be a λ_2-optimal solution, where $\lambda_1 > \lambda_2$. Then $S_1 \subseteq S_2$.*

Proof. We distinguish four classes of polygons: those that are neither in S_1 nor in S_2; those that are only in S_1; those that are only in S_2; and those that are in $S_1 \cap S_2$. We denote these classes by $T_0 = S \setminus (S_1 \cup S_2)$; $T_1 = S_1 \setminus S_2$; $T_2 = S_2 \setminus S_1$; and $T_3 = S_1 \cap S_2$. By A_i we denote the area of T_i, and by L_{ij} we denote the total length of the edges that are shared by T_i and T_j; see Figure 4.4. Let λ_i' be $1 - \lambda_i$.

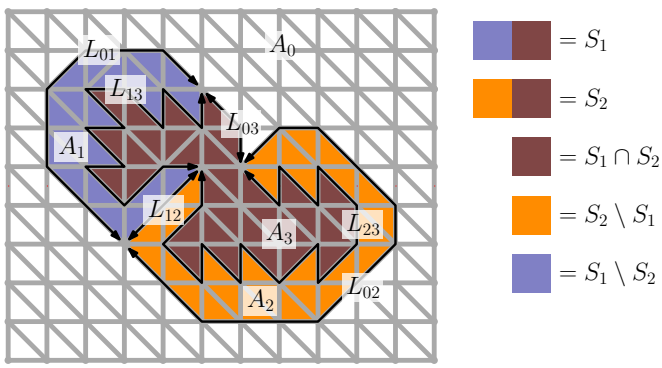


Figure 4.4: Schematic visualization of two λ-optimal solutions S_1 and S_2. The labels A_i refer to the areas and the labels L_{ij} refer to the line lengths used in the proof of Theorem 2.

The λ_1-optimal solution $S_1 = T_1 \cup T_3$ is at least as good as $S_1 \cap S_2 = T_3$ for $\lambda = \lambda_1$:

$$\begin{aligned} & \lambda_1(A_1 + A_3) + \lambda_1'(L_{01} + L_{03} + L_{12} + L_{23}) \\ \leq \quad & \lambda_1 A_3 + \lambda_1'(L_{03} + L_{13} + L_{23}) \\ \Leftrightarrow \quad & \lambda_1 A_1 + \lambda_1' L_{12} \leq \lambda_1'(L_{13} - L_{01}). \end{aligned} \tag{4.2}$$

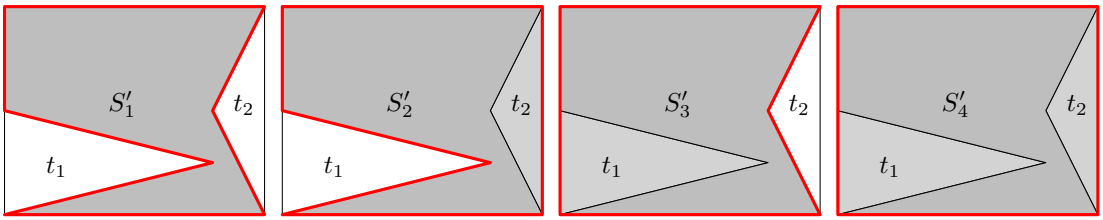
The λ_2-optimal solution $S_2 = T_2 \cup T_3$ is not worse than $S_1 \cup S_2 = T_1 \cup T_2 \cup T_3$ for $\lambda = \lambda_2$:

$$\begin{aligned} & \lambda_2(A_2 + A_3) + \lambda_2'(L_{02} + L_{03} + L_{12} + L_{13}) \\ \leq \quad & \lambda_2(A_1 + A_2 + A_3) + \lambda_2'(L_{01} + L_{02} + L_{03}) \\ \Leftrightarrow \quad & \lambda_2'(L_{13} - L_{01}) \leq \lambda_2 A_1 - \lambda_2' L_{12}. \end{aligned} \tag{4.3}$$

Now suppose $S_1 \nsubseteq S_2$, that is, $A_1 > 0$. With $\lambda_1 > \lambda_2 \geq 0$, Equation Equation (4.2) then implies $L_{13} - L_{01} > 0$. Since $\lambda_2 < \lambda_1$, we also have $\lambda'_1 < \lambda'_2$. Combining this with Equation (4.2) and Equation (4.3) gives:

$$\lambda_1 A_1 \leq \lambda_1 A_1 + \lambda'_1 L_{12} \leq \lambda'_1 (L_{13} - L_{01}) < \lambda'_2 (L_{13} - L_{01}) \leq \lambda_2 A_1 - \lambda'_2 L_{12} \leq \lambda_2 A_1.$$

Thus, $\lambda_1 A_1 < \lambda_2 A_1$. However, this contradicts the assumption $\lambda_2 < \lambda_1$. Therefore, the assumption $S_1 \nsubseteq S_2$ must have been false, and we conclude $S_1 \subseteq S_2$. $\square$



(a) no selection (b) t_2 selected (c) t_1 selected (d) t_1 & t_2 selected

Figure 4.5: Polygon with triangulation of the exterior space. Selecting none, one or all triangles leads to all four possible solutions. The given polygon (set B, gray) must always be selected; red (bold) edges delineate a solution S'.

Example for Theorem 2. We discuss the proven Theorem 2 for a polygon with two triangular concavities, which is shown in Figure 4.5. The concavities can be filled by adding two triangles t_1 and t_2, which leads to four different solutions. All of them are Pareto optimal solutions and only one corresponds to a nonextreme nondominated point (Figure 4.6). As a consequence, when considering only λ-optimal solutions, the triangle t_1 can only be selected if t_2 is selected as well.

We can conclude from the example and the theorem that our solution set has a hierarchical structure. This follows directly from the fact that solutions with small areas are contained in those with larger areas. As a result, there can be at most as many solutions as there are polygons in S. Thus, the size of the solution set is $O(n)$, where n is the number of polygons in S.

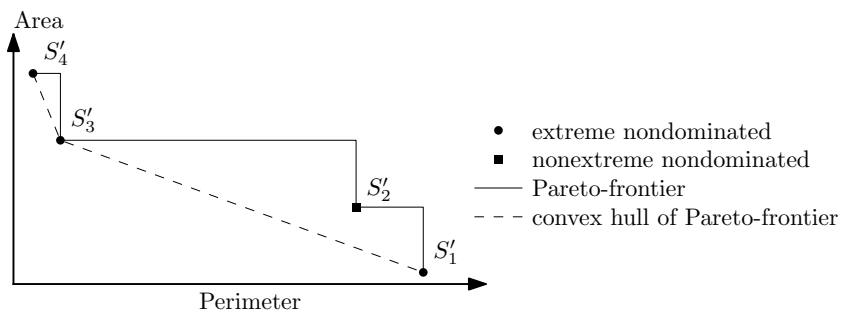


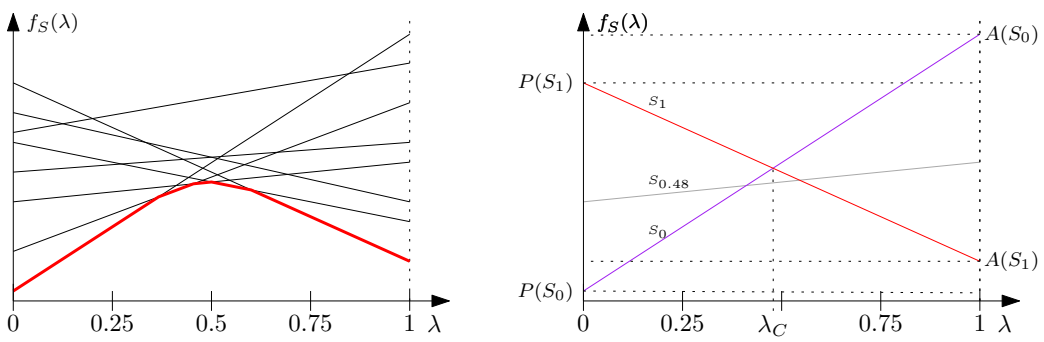
Figure 4.6: Set of solutions for the instance in Figure 4.5 with respect to area and perimeter.

Computing the solution set. According to the discussion so far, a set of linear size exists that contains a λ-optimal solution for every $\lambda \in [0, 1]$. We now discuss how to compute such a set efficiently. We do this with a classical method for multi-objective optimization (Cohon, 1978), which in the literature is sometimes referred to as *dichotomic scheme* (Przybylski et al., 2010) or *chord algorithm* (Daskalakis et al., 2016). For a given instance S, B of Problem 2, a search window $[\lambda_{\mathrm{L}}, \lambda_{\mathrm{U}}]$, and a parameter $\varepsilon \geq 0$, we aim to compute a set containing for every $\lambda \in [\lambda_{\mathrm{L}}, \lambda_{\mathrm{U}}]$ a solution that is at most factor $(1 + \varepsilon)$ worse than a λ-optimal solution. We set the search window to $[0, 1]$ and $\varepsilon = 0$ in case we want to solve Problem 2. By setting the search window to $[0, 1]$ and choosing some $\varepsilon > 0$, we can approximate the result set of Problem 2 with few solutions. This aim is achieved with the recursive algorithm Approx that is presented in pseudocode in Algorithm 1. In addition to S, B, λ_{L}, λ_{U}, and ε, it requires a λ_{L}-optimal solution and a λ_{U}-optimal solution as input, which can be computed using the graph-cut approach presented in Section 4.2.1. To discuss how the algorithm works, we introduce a geometrical representation of the solutions, which is illustrated in Figure 4.7. For every solution, we plot the objective value as a function of λ. This results in multiple line segments. Our goal is to determine the lower envelope of those line segments, which is highlighted in Figure 4.7a. Each line that is part of the lower envelope is part of our solution set as it has the lowest objective value for a specific λ. Initially, two solutions are known, which are optimal for λ_{L} and λ_{U}. For the discussion we assume $\lambda_{\mathrm{L}} = 0$ and $\lambda_{\mathrm{U}} = 1$.

The algorithm, which we call Approx and provide in pseudocode in Algorithm 1, works as follows.

1. Compute λ-optimal solutions S_{L} for λ_{L} and S_{U} for λ_{U} using the min-cut algorithm.

2. Invoke Approx with λ_{L} and λ_{U}: $\quad$ Approx$(S, B, \lambda_{\mathrm{L}}, \lambda_{\mathrm{U}}, S_{\mathrm{L}}, S_{\mathrm{U}}, \varepsilon)$



(a) lower envelope (highlighted red)

(b) first iteration of Approx

Figure 4.7: Geometrical representation of solutions. The objective value of each solution is displayed as a function of λ.

The solutions S_{L} and S_{U} are passed over to the procedure only to allow it to access their associated areas $A(S_{\mathrm{L}})$ and $A(S_{\mathrm{U}})$ as well as perimeters $P(S_{\mathrm{L}})$ and $P(S_{\mathrm{U}})$. We show the geometric representation of the first iteration with the search window of $[0, 1]$ in Figure 4.7b. The solution $S_{\mathrm{L}} = S_0$ refers to $\lambda_{\mathrm{L}} = 0$. The objective value of S_0 at $\lambda = 0$ equals the perimeter of the solution while the objective value at $\lambda = 1$ equals the area of the solution, respectively. The same holds for solution S_1. To keep the presentation compact, we refer to $f_S(0) = A(S)$ for the area and $f_S(1) = P(S)$ for the perimeter of a solution S, respectively.

We outline the further steps of the algorithm in the following. If the solutions S_{L} and S_{U} are equal, we stop the algorithm and report no additional solution. If the solutions are different, as shown with S_0 and S_1 in Figure 4.7b, we determine the intersection point with $\lambda = \lambda_{\mathrm{C}}$ of the two objective functions. If there is any additional optimal solution, it has a lower objective value $f_S(\lambda_{\mathrm{C}})$ than $f_{S_{\mathrm{L}}}(\lambda_{\mathrm{C}}) = f_{S_{\mathrm{U}}}(\lambda_{\mathrm{C}})$.

We compute the λ-value of the intersection as $\lambda_{\mathrm{C}} = (P_{\mathrm{U}} - P_{\mathrm{L}})/(A_{\mathrm{L}} - A_{\mathrm{U}} + P_{\mathrm{U}} - P_{\mathrm{L}})$ for which $f_{S_{\mathrm{L}}}(\lambda_{\mathrm{C}}) = f_{S_{\mathrm{U}}}(\lambda_{\mathrm{C}})$ holds. We obtain the optimal solution for the specific λ_{C} with our min-cut approach. This results in the solution S_{C}. If the objective value of our current solution $f_{S_{\mathrm{L}}(\lambda_{\mathrm{C}})}$ is more than $(1 + \varepsilon)$ times the objective value of the optimal solution $f_{\lambda_{\mathrm{C}}}(\lambda_{\mathrm{C}})$, our approximation error is exceeded. In our displayed example, the intersection point is at $\lambda_{\mathrm{C}} = 0.48$, hence, we obtain the solution $S_{0.48}$. By recursively calling Approx with the ranges $[\lambda_0, \lambda_{0.48}]$ and $[\lambda_{0.48}, \lambda_1]$, we can identify further solutions. This results in the complete lower envelope of the solution set.

As a result, setting $\varepsilon = 0$ leads to the full lower envelope where each line segment of the envelope is λ-optimal. These k line segments correspond to a set of solutions $S_1, \ldots, S_k$. Choosing some $\varepsilon > 0$ leads to an approximated solution set that contains for each $\lambda \in [0, 1]$ a solution that is at most a factor $(1 + \varepsilon)$ worse than optimal. We stop the recursion early if the objective value $f_{S_{\mathrm{L}}}(\lambda_{\mathrm{C}})$ of the current solution S_{L} is less than $(1 + \varepsilon)$-times worse than $f_{S_{\mathrm{C}}}(\lambda_{\mathrm{C}})$. The described process is outlined by Algorithm 1.

Regarding the running time, the number of line segments of the lower hull k is bounded by $O(n)$ due to the linear-size solution set. Since we make two calls of Approx for each solution found, the min-cut algorithm is invoked at most $O(n)$ times. This is also applicable in case $\varepsilon > 0$ is used.

4.3 Experiments

To evaluate our algorithms, we implemented them in Java and conducted tests with building footprints from OpenStreetMap contributors (2020). We transformed the data to UTM coordinates to accurately calculate areas and perimeters

Algorithm 1 Approximation of set of extreme nondominated points

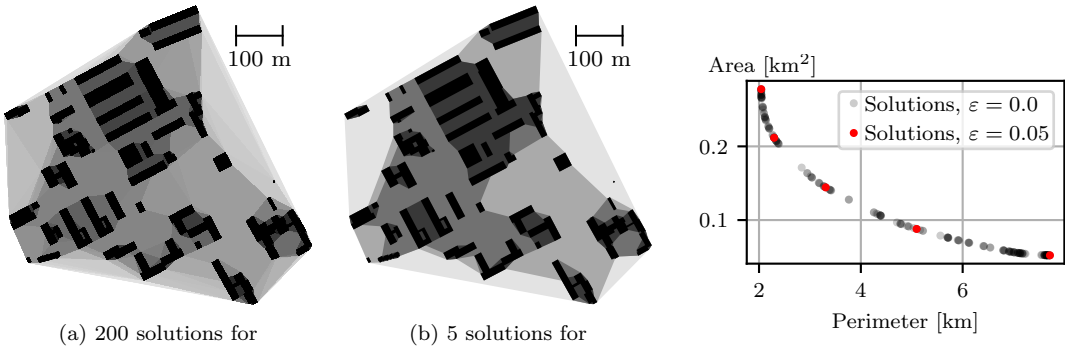
Input: Polygon sets S, B, lower λ (λ_{L}), upper λ (λ_{U}), λ-optimal solutions $S_{\mathrm{L}}, S_{\mathrm{U}}$ for $\lambda_{\mathrm{L}}, \lambda_{\mathrm{U}}$, maximal error ε

Output: Report λ-optimal solutions with $\lambda_{\mathrm{L}} < \lambda < \lambda_{\mathrm{U}}$

```
procedure APPROX(S, B, λ_L, λ_U, S_L, S_U, ε)
    if A(S_L) = A(S_U) and P(S_L) = P(S_U) then return
    end if
    λ_C ← crossing(S_L, S_U)                ▷ Compute λ_C with f_{λ_C}(S_L) = f_{λ_C}(S_U)
    S_C ← mincut(S, B, λ_C)                 ▷ Solve Problem 1 via graph cut
    if f_{λ_C}(S_L) > (1 + ε) · f_{λ_C}(S_C) then    ▷ Maximal error exceeded
        APPROX(S, B, λ_L, λ_C, S_L, S_C, ε)
        Report S_C
        APPROX(S, B, λ_C, λ_U, S_C, S_U, ε)
    end if
end procedure
```

of polygons with a metric unit of measurement. For visualizing the effect of the parameter λ, we present figures where optimal solutions for different values of λ are shown with different gray values together with the input polygons. The same solutions are presented in an online supplement using a separate figure for each value of λ.

Test instance. To provide an overview, we first discuss a test with a smaller instance of 57 buildings; see Figure 4.8. The triangulation for this instance contains 546 triangles. With this test we rather generally tried to get an idea of the kind of polygon groups and aggregated polygons that our algorithms produce. Using the recursive algorithm to compute a set of solutions which contains a λ-optimal solution for every λ (i.e., $\varepsilon = 0.0$) we obtained 200 different solutions; see Figure 4.8a. Some of these solutions differ by only single triangles and are hardly distinguishable in the visualization. When using the algorithm to approximate the result set with $\varepsilon = 0.05$, we obtained only 5 different solutions that are substantially different from each other; see Figure 4.8b. One can clearly perceive the hierarchical structure of the different solutions stated in Theorem 2. Plotting the solutions by their associated perimeters and areas yields a diagram with the extreme nondominated points; see Figure 4.8c. We observe that these points occur at irregular distances, meaning that clusters but also larger gaps exist. Nevertheless, our approximation approach yields a small and representative set of solutions. Due to this result, we decided to focus on approximated result sets in the further experiments, which we present in the following.



(a) 200 solutions for $\varepsilon = 0.0$

(b) 5 solutions for $\varepsilon = 0.05$

(c) areas and perimeters for solutions

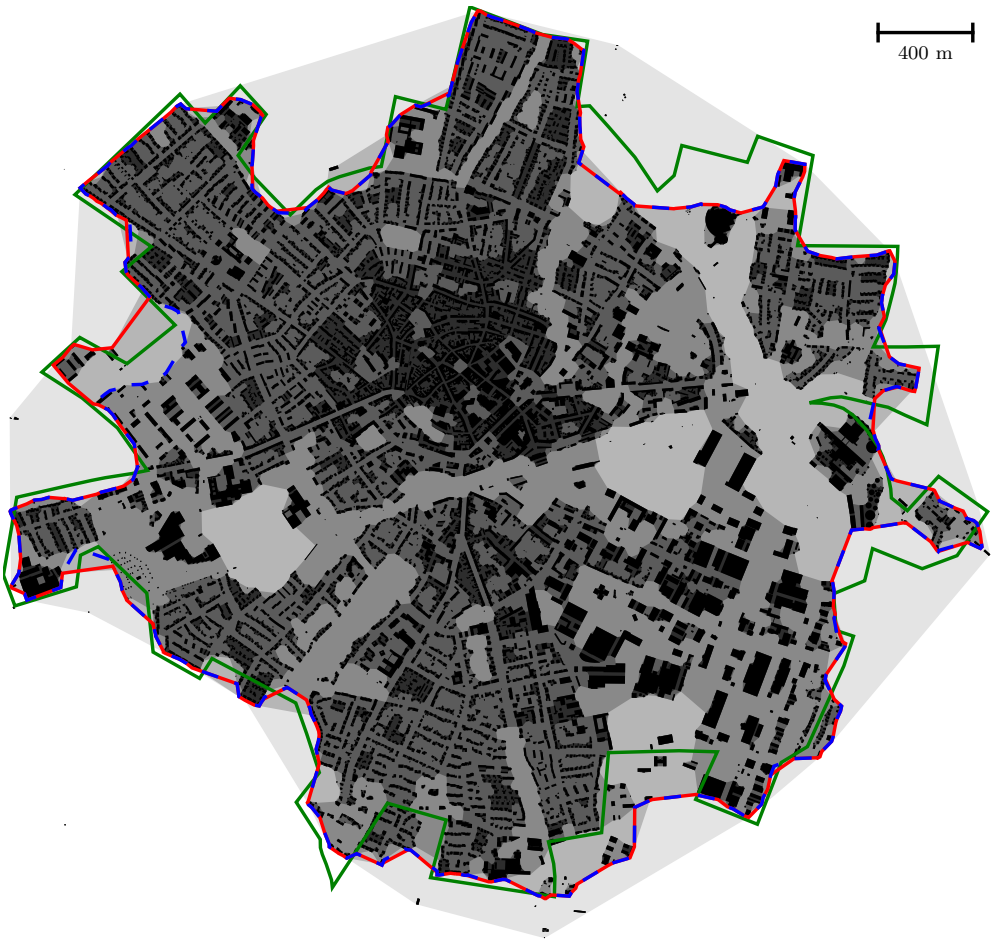
Figure 4.8: Solutions obtained with Algorithm 1 for the same instance without (a) and with approximation (b) and their values for the two objectives (c). Darker polygons correspond to solutions for higher λ values, which consist of multiple groups.

Evaluation data sets. We conducted further experiments with the aim to aggregate building polygons to settlement areas as defined in the digital landscape model DLM250 of the German "Authoritative Topographic-Cartographic Information System" (ATKIS, www.atkis.de). This corresponds to a map scale 1 : 250 000. We evaluate our approach on 99 instances of different sizes, each corresponding to at least one settlement polygon in DLM250, but also including buildings outside the settlement or even multiple settlement areas; see Figure 4.11. The instance sizes vary from 202 to 16999 buildings. They correspond to several German settlement areas as well as groups of settlement areas. These settlement areas are selected from different regions and population densities to have variety in our test cases. We first consider two cases in more detail, namely the settlement areas of Euskirchen and Ahrem, which consist of 16999 and 859 input polygons, respectively. Their corresponding bicriteria shapes are shown in Figures 4.9a and 4.9b.

Evaluation metrics. To evaluate our aggregation results, we compute a rather accurate approximation for our test examples with $\varepsilon = 10^{-6}$. By choosing this small but non-zero error tolerance we were able, in particular, to avoid computing a large set of rather uninteresting solutions in which only few small triangles are added to the input polygons. For visualization purposes we stick to $\varepsilon = 0.1$. We then compared every solution in the result set with the corresponding settlement polygon in the DLM250. For comparing a solution S_1 with the reference S_2 we used the following three different metrics:

- The *Jaccard index*, which is also known as *Intersection over Union* (IoU), for polygons:

$$IoU = \frac{A(S_1 \cap S_2)}{A(S_1 \cup S_2)}$$



(a) Euskirchen (16999 input polygons)

(b) Ahrem (859 input polygons)

Figure 4.9: Result set of two evaluation data sets with $\varepsilon = 0.1$. The green outline corresponds to the ground-truth polygon. The red and blue outlines, which are mostly the same, represent the best bicriteria shapes in terms of IoU and d_{H}, respectively.

- The *Area Similarity*, as suggested by Podolskaya et al. (2007) for quality assessment of polygon generalization:

$$V_{\mathrm{A}} = 1 - \frac{|A(S_1) - A(S_2)|}{\max\{A(S_1), A(S_2)\}}$$

- The *discrete Hausdorff distance* d_{H} of the polygons' boundaries, whose vertex sets we denote as $V(S_1)$ and $V(S_2)$. It adds to the other three measures that it indicates the maximum difference of the solutions with respect to the Euclidean distance d.

$$d_{\mathrm{H}} = \max\left\{ \max_{v_1 \in V(S_1)} \left\{ \min_{v_2 \in V(S_2)} \{d(v_1, v_2)\} \right\}, \max_{v_2 \in V(S_2)} \left\{ \min_{v_1 \in V(S_1)} \{d(v_1, v_2)\} \right\} \right\}$$

The similarity measures IoU and V_{A} are in the range of $[0, 1]$, where 0 represents minimum similarity and 1 represents maximum similarity. For cases where one polygon is contained in a second polygon, V_{A} equals IoU.

An important selection criterion defined in the specifications of the DLM250 is that all settlement areas must be larger than 40 hectares (Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV), 2022). Therefore, we removed all polygons smaller than this threshold from a solution before computing the three measures. The resulting measures for the settlement area of Euskirchen are shown in Figure 4.10. For all $\lambda > 0.105$ the solution contained no contiguous polygon reaching the 40 hectares threshold, meaning that no similarity can be found. The diagram reveals a correlation of IoU and Area Similarity V_{A}; in fact they are equal for $\lambda > 0.05$.

For each metric we identified the value for λ that yields the solution of maximum similarity or minimum Hausdorff distance. For the cases where IoU, V_{A}, and Hausdorff distance have their respective best values, the other metrics are similar to their best values. The solutions of minimum Hausdorff distance and maximum IoU are also depicted in Figure 4.9 as blue and red lines, respectively.

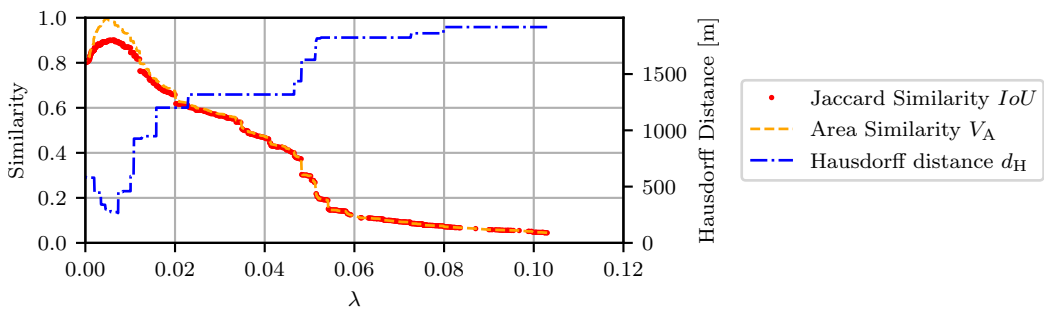


Figure 4.10: Evaluation metrics for the settlement area of Euskirchen. For $\lambda > 0.105$ all polygons of the result set are below the specified minimum area.

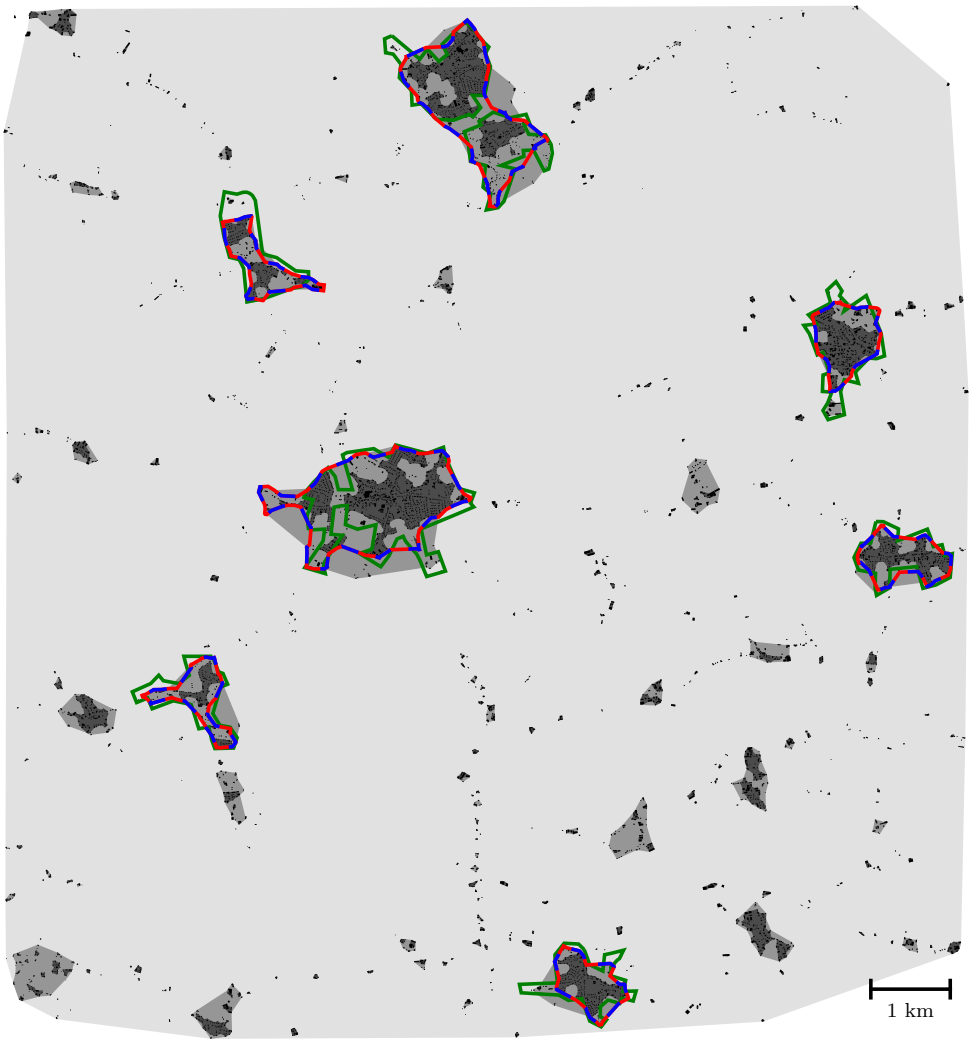


Figure 4.11: A large instance aggregated by our approach showing the separation capabilities of our method. Our approach recognizes all settlement areas recorded in the ground truth data as well as several smaller ones. The latter could be filtered out with a size threshold.

We observe that the shapes are indeed similar, but that the boundary of the reference solution contains more angles close to 90°. Using a schematization algorithm in a post-processing step might increase the similarity in this respect.

Detecting multiple settlement areas. The instances of Ahrem and Euskirchen are limited to a single settlement area. Using our approach on a larger instance, we demonstrate our ability to separate different settlement areas into multiple regions. For this reason, we run our approach on the instance shown in Figure 4.11. The detected groups are in line with the groups of the ground-truth data. This shows that a single λ can determine multiple groups without changing the parameter. We included 12 group instances in our evaluation in Figure 4.12, while the individual settlement areas of these groups are not evaluated individually.

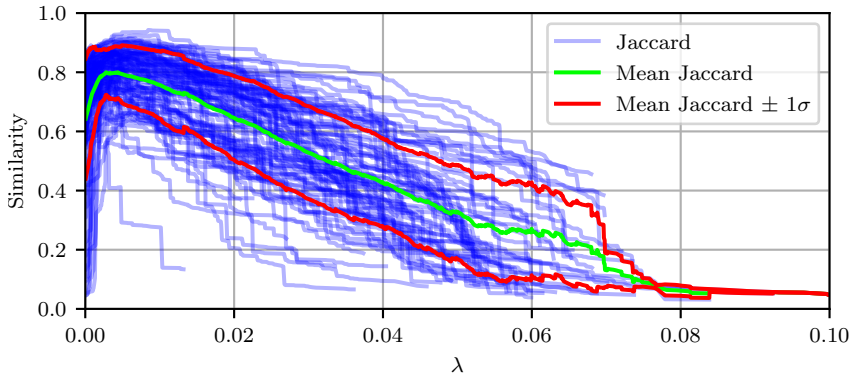


Figure 4.12: IoU-scores of multiple instances in relation to our λ-value. Each blue line represents an independent instance.

Implementation details. In our Java implementation, we used the conforming Delaunay triangulation algorithm implemented in the library JTS Topology Suite (JTS)[1] This algorithm can be implemented to run in the worst case in $O(nm \log(n^2/m))$ time on a graph with n nodes and m edges (Goldberg and Tarjan, 1988), but the JGraphT documentation reports a worst-case running time of $O(n^3)$. This implies that our implementation of the algorithm for Problem 1 runs in $O(n^3)$ time, where n is the number of polygons in the set S. This is substantially higher than the sub-quadratic running time that is achievable according to Theorem 1, but sufficiently low to solve problem instances as the ones we discussed above. Our implementation of the recursive algorithm for Problem 2 uses parallelization for the two recursive calls, which turned out to improve the running time substantially. In particular, the running time improves from 42 seconds to 8 seconds for $\varepsilon = 10^{-6}$ (448 solutions) on the instance of Ahrem by using parallelization on an AMD Ryzen 7950X with 32 threads. Note that for the application in interactive maps, the algorithm would need to be run only once to set up a data structure storing the solutions (Timpf and Frank, 1995). This data structure can be repeatedly queried to obtain representations at different scales.

4.3.1 Choosing the Parameter λ with the Help of Reference Solutions

After taking a closer look at our approach using two instances, we will explore the generalization capabilities of the parameter λ on more datasets. Our goal is to show that for a fixed map scale there exists a certain λ which provides reasonably good results for a wide range of cases. For this evaluation, we focus on the IoU score, as this has been proven to be meaningful in the evaluation so far. Using

[1] https://github.com/locationtech/jts/releases/tag/jts-1.15.1. For computing the graph cuts, we used the Push Relabel algorithm implemented in the library JGraphT (Michail et al., 2020).

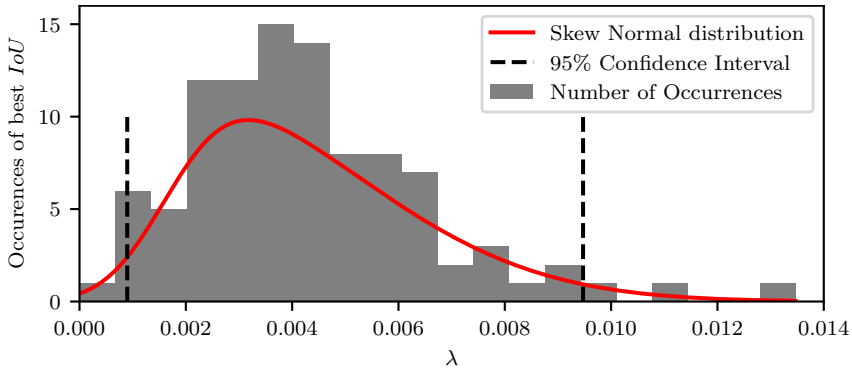


Figure 4.13: Histogram of the λ values of the respective best solutions with respect to the IoU value.

this metric, we now plot our 99 instances in a graph comparable to Figure 4.10. In addition to the IoU scores, we also visualize the mean and the 1σ interval in Figure 4.12 to estimate the variance. Analyzing all instances results in the best mean IoU score of 0.7997 ± 0.0759 for a λ of 0.0028. In addition to the general overview, we present the best IoU-score of each individual test case in a histogram; see Figure 4.13. To smooth out discontinuities of the histogram, we also fitted a skew-normal distribution to it, whose maximum is at $\lambda = 0.0035$ and whose mean is $\lambda = 0.0043$. We can determine the 95%-interval $[0.0009, 0.0095]$ by using the standard deviation $\sigma = 0.0022$ and Fisher's skew of 0.7495. The best scores of Ahrem and Euskirchen with respect to the IoU-score are even included by the 68%-interval $[0.0022, 0.0066]$ with λ-values of $\lambda = 0.0041$ and $\lambda = 0.0064$, respectively.

In the application, these findings can be used for restricting the search range of λ. In our example, only the solutions for $\lambda \in [0.0009, 0.0095]$ need to be computed, as other values for λ do not correspond to the required aggregation level. For the dataset of Ahrem, using this search range and $\varepsilon = 0.01$ produces only three solutions, from which the best can subsequently be selected.

4.3.2 Edge-Aligned Polygons

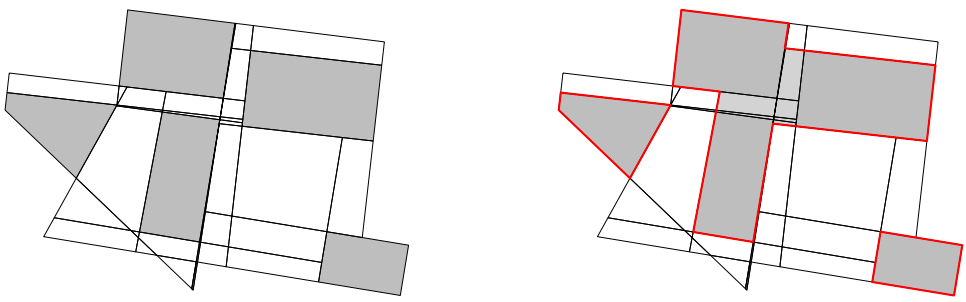
While so far we used our method with a conforming Delaunay triangulation as the underlying planar partition, other types of planar partitions can be used as well. We can use this flexibility of our method to preserve the edge orientations of the given polygons, which is desirable in the context of building generalization (Haunert and Wolff, 2008). More precisely, we generate the planar partition by linearly extending the edges of the input polygons.

Constructing the planar subdivision. In the first step of our approach, we consider each individual polygon $p \in B$ and process the outer and inner rings.

For every vertex v we check whether it forms a convex corner of the polygon by inspecting the two adjacent vertices. If this condition holds true, the incident edges are linearly extended at v until they hit an input polygon or the bounding box of the whole set B of input polygons. For an efficient implementation we store the polygons in a Sort-Tile-Recursive (STR) packed R-tree (Rigaux et al., 2002). The linearly extended polygon edges yield a line arrangement, whose faces constitute the set S of input polygons for our method. After the generation of the planar partition we apply our algorithm in the same way as when using a triangulation.

Running time. Note that when using this method the size of S is in $O(m^2)$, where m is the total number of vertices of the polygons in B. Consequently, the worst-case running time for solving Problem 1 is $O(m^4/\log m)$. Using the graph-cut implementation of JGraphT this increases to $O(m^6)$. Nevertheless, the time complexity $O(n^2/\log n)$ stated in Theorem 1 is still correct, since n refers to the number of simple polygons that form the planar subdivision S.

To keep the running time in practice low, we terminate the extension of the polygon edges when they hit another input polygon. With this, the number of faces of the resulting line arrangement is still manageable. We show the planar partition resulting from this procedure for five input polygons in Figure 4.14a.



(a) input polygons (gray) with extended edges

(b) bicriteria shape for $\lambda = 0.4$

Figure 4.14: Extending the boundary edges for given polygons to create edge-aligned polygons.

Discussion. For $\lambda = 0.4$ we obtain a solution consisting of three aggregated groups composed of one to two input polygons; see Figure 4.14b. With this the edge orientations of the input polygons are preserved. Furthermore, we compute the results of our approach using the edge-aligned polygons and show the results for Ahrem in Figure 4.15. For this instance the solutions produced by selecting the best IoU score and Hausdorff distance are the same. The running time to retrieve the approximated solution set for Ahrem with $\varepsilon = 10^{-6}$ increases from 8 seconds to 176 seconds when using the elongated edges instead of a conforming Delaunay triangulation. This increased running time is a direct consequence of

Figure 4.15: Ahrem aggregated with polygons created by extruding polygon edges.

the increased number of polygons. While the conforming Delaunay triangulation results in 5,280 triangles between the 859 input polygons in the case of Ahrem, the elongated edges result in 76,063 polygons.

4.3.3 Comparison to α-shapes

For further comparisons, we compute bicriteria shapes as well as α-shapes based on the same conforming Delaunay triangulation. The methods differ only in the selection criteria for the triangles. First, we compute the optimal solution of Ahrem using our approach. Then, we select the α-shape that is the best match to our solution when considering the IoU-value. A comparison of both solutions is visualized in Figure 4.16. For this comparison we do not limit the resulting polygons to a given area threshold. This side-by-side view of the two different

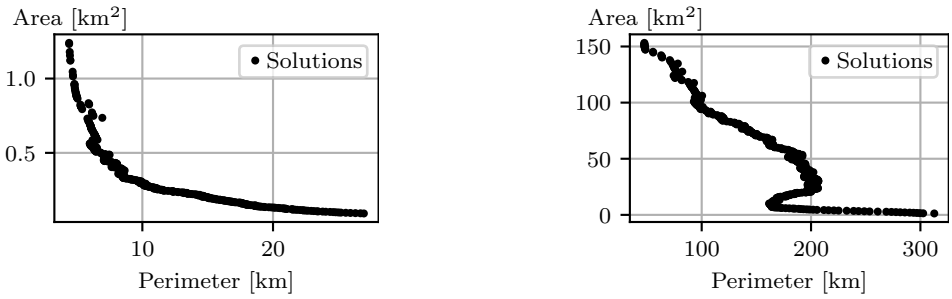
(a) Ahrem aggregated with our approach

(b) Ahrem aggregated with α-shapes

Figure 4.16: Ahrem generated with our proposed approach (left, $\lambda = 0.0058$) and with an α-shape (right, $\alpha = 0.0017$). The α-shape is selected to maximize the IoU-score with respect to the solution of (a).

results shows the advantage of the bicriteria shape over the α-shape. The α-shape leads to long narrow bridges, which can be observed in the lower right and left corner of Figure 4.16b. The narrow but relatively long gaps are closed in case of the α-shape. In contrast, they remain open with our approach, because filling these gaps neither reduces the area nor the perimeter, which is shown in Figure 4.16a.

This observation that α-shapes are not λ-optimal is also shown in Figure 4.17a and Figure 4.17b. The set of points representing the α-shapes for all values of α does not constitute a convex curve. This means that not all α-shapes correspond to extreme nondominated points. Moreover, when continuously increasing α and keeping track of the α-shape, one does not obtain a sequence of solutions with monotonically decreasing perimeter. This is particularly evident in Figure 4.17b. Hence, other than the parameter λ of our method, the parameter α of α-shapes does not control the degree of simplification of the input data with respect to the perimeter criterion.



(a) Ahrem

(b) large instance (Figure 4.11)

Figure 4.17: All α-shape solutions of instances shown in Figure 4.16 and Figure 4.11.

4.4 Conclusion

We have presented efficient algorithms for polygon aggregation optimizing a balance between a small total area and a short total perimeter of the output polygons. We combined the two criteria in a weighted sum, which we parameterized with a single parameter $\lambda \in [0, 1]$. The first problem we studied asked for a single optimal solution for a fixed λ. It turned out that this problem can be solved by computing a minimum cut in a graph. A second problem asked for an output set containing an optimal solution for every possible value for λ. We showed that a linear-size hierarchically nested set with the requested property can be efficiently computed with a recursive algorithm that uses the graph-cut algorithm as a subroutine. Moreover, we showed how to approximate such a set using the same recursive algorithm.

Our experiments showed that the computation of bicriteria shapes is fast enough to process realistic problem instances, although we did not use the fastest known (i.e., sub-quadratic) graph-cut algorithm in our implementation. We consider it astonishing how few solutions were needed to approximate a set containing an optimal solution for every λ: For our largest instance with 16999 building footprints, a set of six solutions sufficed to include for every λ a solution that is at most 10% worse than optimal. Since the number of graph cuts computed by the recursive algorithm is in the order of the size of its output set, the approximation for the above-mentioned instance was achieved relatively fast, in 13.9 seconds. Our experiments support the claim that bicriteria shapes can aggregate building footprints to polygons that are quite similar to settlement areas as given in an official topographic database of scale $1:250\,000$. In addition, our experiments on a larger dataset assist the finding that the parameter λ is within a narrow search window when computing a solution close to the reference solution considering a target map scale. We extend the experiments by replacing the triangulation underlying our method with a planar partition that preserves the edge directions of the given geometry. Finally, our approach exposes the benefit of more compact regions in direct comparison to α-shapes. Moreover, bicriteria shapes show a monotonically increasing perimeter and a monotonically decreasing area for increasing values of the parameter λ, whereas for α-shapes the perimeter does not behave monotonically.

As an idea for future research it would be interesting to consider relaxed or more constrained versions of the aggregation problem. For example, one could relax the requirement to include every input polygon in the output and/or introduce a hard size constraint for the output polygons. Most importantly, however, we see our work as a step towards multi-criteria optimization in cartography. As next steps one could consider more than two criteria or look at Pareto-optimal solutions rather than just at the extreme nondominated (i.e., λ-optimal) solutions. In order to optimize the processing times, it would be beneficial to utilize the nested structure of the solution. In this context, a subsequent evaluation of the running times for different sizes of datasets would be interesting.

Chapter 5

Grid-Based Euler Diagrams

This chapter is based upon a joint publication with Markus Wallinger, Annika Bonerath, Sven Gedicke, Martin Nöllenburg and Jan-Henrik Haunert (Rottmann et al., 2023). The user study was developed, conducted and written by Annika Bonerath and Sven Gedicke. The development and write up of different rendering techniques was done by Markus Wallinger and Martin Nöllenburg. With the supervision of Jan-Henrik Haunert, I (Peter Rottmann) developed the integer linear programming model and wrote the remaining parts of the publication, i.e., the introduction, the explanation of the ILP, the quantitative evaluation and the conclusion.

Visualizing sets of elements and their relations is an important research area in information visualization. In this chapter, we present *MosaicSets*: a novel approach to create Euler-like diagrams from non-spatial set systems such that each element occupies one cell of a regular hexagonal or square grid. The main challenge is to find an assignment of the set elements to the grid cells such that each set constitutes a contiguous region. As use case, we consider the research groups of a university faculty as elements, and the departments and joint research projects as sets. We aim at finding a suitable mapping between the research groups and the grid cells such that the department structure forms a base map layout. Our objectives are to optimize both the compactness of the entirety of all cells and of each set by itself. We show that computing the mapping is NP-hard. However, we can solve real-world instances optimally within a few seconds using integer linear programming. Moreover, we propose a relaxation of the contiguity requirement to visualize otherwise non-embeddable set systems. We present and discuss different rendering styles for the set overlays. Our evaluation is based on a case study with real-world data and comprises quantitative measures as well as expert interviews.

5.1 Introduction

Set visualization is an important branch of information visualization that generally deals with the visualization of set systems, i.e., the relationships between multiple sets (e.g., set intersections, inclusions, exclusions) and of individual elements with their set memberships. As a concrete use case and our running example, we consider the scenario that the board of an institution (e.g., a university faculty) aims to improve its strategic planning processes by supporting the discussions at meetings with informative visualizations. In particular, the institution board wishes to visualize the institution's partition into organizational units (e.g., departments and research groups) as well as important intra-institutional collaborations (e.g., larger research projects involving multiple research groups of different departments). In this scenario, the research groups are the elements of the set system and the departments as well as the intra-institutional collaborations are the sets. Particularly, each research group belongs to exactly one department and possibly one or more intra-institutional collaborations. Consequently, the departments form a disjoint set cover of all research groups. In MosaicSets, we will exploit this property of having a partition of all elements into a disjoint set cover, which is a property that many real-world set systems have.

For visualizing abstract set systems, Euler and Venn diagrams are commonly used and several methods for computing them have been proposed. However, they usually focus on the set relationships and, even in area-proportional diagrams, either do not show individual elements at all or they do not play a primary role in the diagrams. However, in our use case, elements such as the research groups are of primary importance for the intended tasks, and therefore it is desirable that each element is prominently represented with an area of the same size and shape, ensuring an equitable visual representation of each research group. Since the existing Euler-like methods do not satisfy this requirement, we present a new approach that embeds a given set system into a prescribed grid, which for example may consist of square-shaped or hexagonal tiles, such that each element occupies exactly one grid tile (or grid cell). Our primary goal is to ensure that for each of the given sets the corresponding grid cells constitute a contiguous region as in an Euler diagram. Coloring the cells according to the partition into a disjoint set cover, e.g., by organizational unit of the assigned element, we obtain a visualization similar to a grid map (Eppstein et al., 2015) with uniform cells and schematic region boundaries; see Figure 5.1. This serves as the base map of our set system. Sets representing intra-institutional collaborations can then be visualized as contiguous overlays on top of the base map, for example, using colored contours or visual links as Kelp diagrams (Dinkla et al., 2012; Meulemans et al., 2013). If the number of such collaborations is small, they all may be visualized at the same

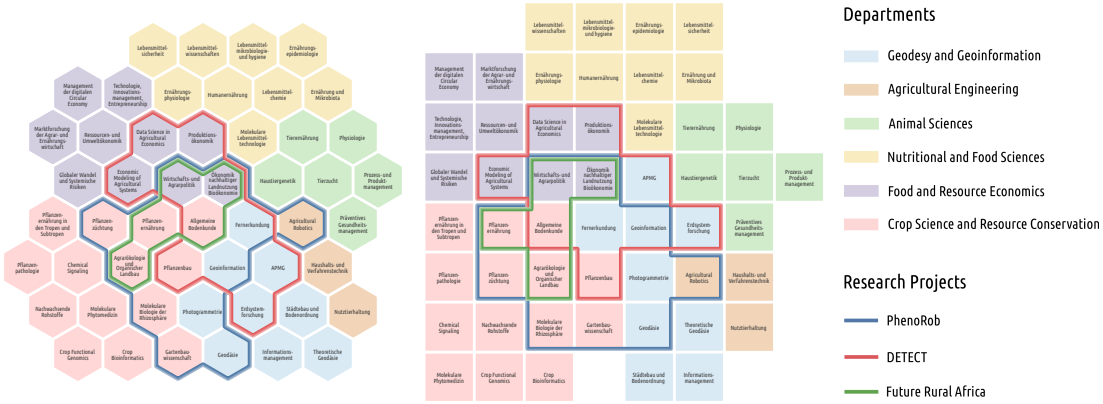


Figure 5.1: Visualizing the research groups of the Agricultural Faculty of the University of Bonn with *MosaicSets* using the model variant MSE optimizing eccentricity-based compactness of sets (see Section 5.6.3).

time. However, even if at any time only one or few selected collaborations are shown, we consider it beneficial to compute a single base map embedding once under consideration of all collaborations, which can then be shown on demand in an interactive setting or using a small-multiples approach. That way the grid map of organizational units provides a stable context for visualizations of different collaborations.

Contribution and outline. Grid maps have been introduced primarily for the purpose of associating disjoint geographic regions (e.g., administrative regions) with cells or tiles of a regular grid. In contrast, we here use the map as a metaphor to visualize non-geographical data, namely an abstract set system, in an intuitive way. While methods for the computation of map-like visualizations of graphs (Gansner et al., 2009a) and set systems (Efrat et al., 2015) have been presented before, using grid maps for set visualization is new. Consequently, with this chapter we introduce *MosaicSets* as a new set visualization technique. We aim to demonstrate the potential of MosaicSets for the described use case but also to open up a new topic of research for the network and set visualization community. As concrete contributions, we present

- our design decisions and a corresponding formal mathematical model for optimizing the MosaicSets visualization as a constrained hypergraph embedding problem, alongside with a proof of its NP-completeness (Section 5.3),
- an exact method for solving the hypergraph embedding problem using integer linear programming including variations to optimize compactness or relax contiguity requirements if needed (Section 5.4),
- an implementation providing a choice of square or hexagonal grids for the base map and two overlay rendering styles for highlighting the additional sets (Section 5.5), and

- a qualitative evaluation based on an expert interview for the real use case, which has inspired this work: the strategy development of the Agricultural Faculty of the University of Bonn; this is complemented by quantitative computational experiments, which explore the performance and resulting quality of MosaicSets (Section 5.6).

We invite the reader to explore our two interactive visualizations under `https://www.geoinfo.uni-bonn.de/mosaicsets` and our code at `https://gitlab.igg.uni-bonn.de/geoinfo/mosaicsets`.

5.2 Contextual Background

The use of grid maps for visualizing geographic information is a well-established technique in cartography and geographic information visualization. Before introducing MosaicSets, which creates a map-like visualization of non-spatial data, we will discuss related work on grid maps and other visualization techniques that are visually similar to MosaicSets.

5.2.1 Geographic Information Visualization Techniques

Although in our problem setting of set visualization the data have no spatial component, several geovisualization techniques are visually and computationally similar.

Tree maps. An approach that is visually similar to MosaicMaps are spatial treemaps, which partition a set of geometric elements according to a hierarchical tree structure. Generally this does not lead to regular grid representations. An example for such spatial treemaps is an approach by Jern et al. (2009) that combines treemaps and choropleth maps. Efforts have been undertaken to tweak treemaps to produce regular grid visualizations, e.g., spatially ordered treemaps (Wood and Dykes, 2008), OD-maps (Wood et al., 2010), and spatial matrices (Wood et al., 2011). Our approach differs in two aspects: (i) our elements do not need to be mapped according to a spatial location, (ii) our problem comprises contiguity requirements for the sets.

Visually similar to MosaicSets are also mosaic cartograms (Cano et al., 2015). Here, we are given a political map with an integer weight per region. The aim is to map each region to a set of grid cells in a square or hexagonal grid whose number is proportional to that region's weight. In contrast to MosaicSets, mosaic cartograms shall represent the spatial input in terms of adjacencies, shapes and relative positions well, and they only show a partition of the grid, but no overlapping sets.

Grid maps. Grid maps are an approach that is similar to MosaicSets both visually and in the problem setting. A grid map (also known as tile map (McNeill and Hale, 2017) or equal area unit map (Schiewe, 2021)) associates each area of a geographic subdivision of the plane, e.g., an administrative region, with a cell of a regular grid. Applications of grid maps are, e.g., in news and media (Berkowitz and Gamio, 2015; Radburn, 2016) or in the visualization of research results (Slingsby and van Loon, 2017; Kelly et al., 2013; Wood et al., 2011). From a practical point of view the topic is addressed with a wide range of blog-posts, reports, and web-tools (Shaw, 2016; Wongsuphasawat, 2016; Zachary, 2015). From a computational point of view a main challenge in computing a grid map is to find a suitable mapping between the geographic regions and the grid cells (Schiewe, 2021). Eppstein et al. (2015) presented an approach that tackles this task considering three criteria: preserving (i) location, (ii) adjacencies, and (iii) relative orientation. Due to computational efficiency only criteria (i) and (iii) are considered for the optimization while all three criteria are then used as a metrics for rating the generated grid maps. Meulemans et al. (2016) presented an approach for generating grid maps for small multiples with a special focus on white space handling. McNeill and Hale (2017) published an approach where multiple grid maps are generated, and the user can choose the most suitable one. Later a more complex problem setting was introduced where the geographic regions are partitioned into sets (Meulemans et al., 2021). Now the matching between the geographic regions and cells has to satisfy that the union of cells of one set shall represent the corresponding geographic region well. Especially the latter version, where the subdivisions' elements are partitioned into sets, is similar to our problem setting. Nevertheless, our problem is different, since we can handle set systems with arbitrary set relations, and we do not aim at providing a good geographic representation.

Another related technique was recently presented by Bekos et al. (2022). They discuss the embedding of spatial hypergraphs, where each vertex has a spatial location and each hyperedge can join any number of vertices. They produce an embedding such that each vertex lies on a rectangular or concentric grid. As optimization criteria the vertex displacement should be small, and the embedded edges should be visually clear. Their approach differs in several ways from MosaicSets: (i) their visualization leads to a sparse placement of vertices in a grid, while we aim at a compact representation using cells and (ii) they consider spatial closeness with respect to the given positions in their objective.

We want to emphasize that the challenges in geovisualization differ since the positions of the elements in the visualization should at least coarsely reflect their geographical locations. Hence, applying the presented techniques to our problem setting for non-spatial set systems would not lead to a satisfying solution.

5.2.2 Maps as a Metaphor

As cartographic maps are well known, some visualization methods, including the previously discussed MapSets (Efrat et al., 2015) and also MosaicSets, use this intuitive presentation of information to visualize non-spatial data; see the survey by Hogräfer et al. (2020). GMap (Gansner et al., 2009a,b) is an example using such a map metaphor that is in some sense similar to MosaicSets. Both approaches show non-spatial data. In GMap, the input is a general graph, and we are looking for a set of touching polygons that represent clusters in the data, which can be singletons or sets of closely connected vertices. Thereby the data and their underlying structure shall be represented in a comprehensible and visually appealing way. Other than our approach, GMap does not aim at a regular grid representation and their input is a graph and not a set system.

5.3 Towards a Formalization of MosaicSets

Let us recap the main principle of MosaicSets based on the example in Figure 5.2. The input is a set system, where each set is a department or a project and each element is a research group (or a key value representing it). The output is a map-like visualization where each element occupies a cell of a grid and each set is a contiguous region. In this section, we first discuss the ideas that led to this principle as well as the design goals that we follow when generating the visualization (Section 5.3.1) and then present a formal definition of the problem (Section 5.3.2) as well as a short proof of its NP-completeness (Section 5.3.3).

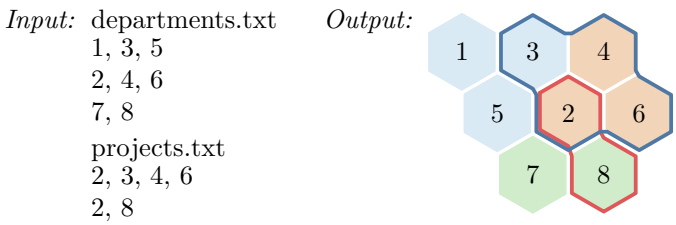


Figure 5.2: Required input files and a solution using a basic rendering style. Additionally, our method requires several parameter values as input.

5.3.1 Design Decisions

The original intention behind MosaicSets is to visualize the structure of an institution (e.g., faculty) broken down into smaller units (e.g., departments and, as atomic elements, research groups) as well as intra-institutional collaborations. The visualization should support the discussions at meetings of the institution's board and its commissions. In many situations, questions of strategic importance

need to be discussed: What are the strengths and weaknesses of the faculty? How large are the different departments, in terms of both absolute and relative sizes? Between which parts of the faculty are the links well established and where are they underdeveloped? For instance, a frequent situation is that the chair of a recruitment commission presents the concept of a new professorship at a faculty board meeting. Then, a visualization should enable the commission chair to explain how the new position will be embedded within the faculty's organizational structure and research networks. Another application that we target is the communication of the faculty's structure to the outside world; e.g., the visualization may be published on the faculty's website or used in meetings with representatives from higher decision levels or funding agencies. The applications of MosaicSets are certainly not limited to academia, since, e.g., companies and non-profit organizations may be faced with similar strategic questions. Generally speaking, the above scenarios suggest creating a clean and intuitive visualization, which can easily be understood by the various stakeholders who are interested and/or already knowledgeable in the data, without a need for extensive training on how to read and use it. We aim for a visually attractive design that invites potential users to engage themselves with the visualization. The visualization should enable the users both to get a quick overview of the data, but also to explore the relationships and patterns between and within the different entities of the depicted organization. Based on this target use case, we emphasize that we do *not* see MosaicSets as an exploratory or analytical tool for big data, which would require additional efforts for data simplification and aggregation.

Map-like visualization. The targeted scenarios require a visualization providing a strategic overview, while also being intuitive and engaging. For this reason, MosaicSets has been designed as a *map-like visualization*: Just like the countries in a political map, each department occupies a dedicated territory. A further essential design decision is to choose an *equitable representation* of each research group: All research groups should be represented with an area of the same size. Requiring a map-like visualization and an equitable representation naturally leads to the use of a *regular tessellation*, which directly entails the advantages of a *schematic map*: low clutter and clear representation of topological relationships. Among the three existing types of regular tessellations, we favor *hexagonal* or *square tessellations* since their tiles are translated copies of each other, which does not hold for triangular tessellations as the third type. Moreover, in a triangular tessellation every tile has only three neighbors, which is a severe limitation when it comes to the embedding of set systems. In square and hexagonal tessellations, on the other hand, there are four respectively six neighbors per tile. These tessellations are commonly used for maps in strategic board

and computer games, which probably helps to understand the visualization as a metaphorical map.

Layout criteria. Next, we discuss the criteria that we use when assigning the elements of the sets to grid tiles. Achieving that the sets are represented as *contiguous regions* is our main goal, since research on human perceptual organization provides evidence that "a connected region of uniform visual properties—such as luminance or lightness, color, texture, motion, and possibly other properties as well—strongly tends to be organized as a single perceptual unit" (Palmer and Rock, 1994). (Note that the term "principle of contiguity" in psychology refers to a different concept, i.e., a close temporal relationship between stimuli and responses that leads to their association (Lachnit, 2006).) Since the contiguity requirement can render the problem infeasible, we will consider problem variants with relaxed versions of it. However, we will not relax the contiguity requirement for the sets corresponding to the organizational units that partition the institution of interest: For example, we strictly enforce that every department of a faculty is represented as a contiguous region, but we tolerate non-contiguous regions for projects in some problem variants. This is because especially the departments should be displayed as territories to give the visualization the look of a political map and because the contiguity requirement can always be satisfied when embedding a partition (if a sufficiently large grid is used). This "political map" can then be used as the base map for visualizing the projects. If the number of projects is small, one may be able to visualize them all at the same time, e.g., by displaying the boundaries of different projects with different colors. However, as this may quickly lead to visual clutter, we also consider displaying the base map with only one project at once. For example, one can use *small multiples* with different versions of the map showing different projects or *interactive maps* allowing the user to select a single project or a few projects for display on top of the base map. Subject to the contiguity requirement, we aim in particular at maximizing the *compactness* both of the visualization as a whole and of the region for each individual set, since according to Gestalt theory humans tend to notice compact forms (Ehrenzweig, 1953) and since maximizing the compactness results in low visual clutter.

5.3.2 Formal Problem Definition

As input, we require a collection $C = \{S_1, \ldots, S_k\}$ of sets and an empty grid map in the form of the adjacency graph $G = (V, E)$ of its cells. We call G the *host graph* and observe that it is planar. Let $S = \bigcup_{i=1}^{k} S_i$. We require $|V| \geq |S|$, i.e., the host graph must be large enough to accommodate all elements in S. With S and C, we obtain a hypergraph $\mathcal{H} = (S, C)$, where S is the vertex set and

C is the set of hyperedges. Now, our basic task is as follows: Find an injection $f\colon S \to V$ such that for $i = 1, \ldots, k$ the subgraph of G induced by $\{f(s) \mid s \in S_i\}$ is connected. Note that f being an injection implies that it maps each element in S to exactly one vertex of G and at most one element to each vertex; hence every element is assigned to one grid cell without any double occupancy. Moreover, the subgraph of G induced by some $V' \subseteq V$ is $G' = G[V'] = (V', E')$ with $E' = \{\{u, v\} \in E \mid u, v \in V'\}$; hence requiring that $\{f(s) \mid s \in S_i\}$ induces a connected subgraph for each $S_i \in C$ is equivalent to the support graph property and hence implies that $G[V']$ for $V' = \{f(s) \mid s \in S\}$ is a planar support graph of $\mathcal{H}$. Furthermore, it ensures that the region for each set S_i is contiguous.

In order to tackle the basic task, we introduce a directed bipartite graph $B = (S \cup V, F)$ with $F = \{(s, v) \mid s \in S \wedge v \in V\}$ that models all possible assignments between elements and grid vertices as directed edges. Selecting an assignment $(s, v) \in F$ thus means setting $f(s) = v$. Finally, to differentiate between different feasible solutions, we introduce a generic cost $w(s, v) \geq 0$ for every possible assignment $(s, v) \in F$. To express the compactness of the whole visualization as a basic optimization objective, we use

$$w(s, v) = \|v.p - \mu\|^2 \ , \tag{5.1}$$

where $v.p \in \mathbb{R}^2$ is the position vector of the center of gravity of the cell represented by v and $\mu \in \mathbb{R}^2$ the position vector of the center of gravity of the set of all cells.

By minimizing the total cost with this setting, allocations of cells close to the center of the visualization are favored, yielding a compact visualization. We will later refine this problem definition to also express that each individual set should be represented with a compact region and to express the compactness of a region based on its perimeter.

5.3.3 Computational Complexity

We can prove that the problem of computing a support of an arbitrary hypergraph that is a subgraph of a planar host graph is NP-complete.

Theorem 3. *For a given planar graph $G = (V, E)$ and a hypergraph $\mathcal{H} = (S, C)$ with $|S| \geq |V|$ it is NP-complete to decide if there is a subgraph of G that is a support graph of $\mathcal{H}$.*

Proof. We reduce from the NP-complete Planar Hamiltonian Circuit problem (Garey et al., 1976), which is defined as follows. Given a planar, cubic, 3-connected graph $G = (V, E)$ determine if G has a Hamiltonian circuit, i.e., a closed path that visits each vertex of V exactly once and returns to its starting point.

It is easy to see that our decision problem belongs to the class NP as verifying whether a given subgraph of G is actually a support of $\mathcal{H}$ can be done in polynomial time. For the hardness reduction, let $G = (V, E)$ be the graph for which we want to find a Hamiltonian circuit and define it as the planar host graph for our hypergraph support problem. Further, we define the hypergraph $\mathcal{H} = (S, C)$ with $S = \{1, 2, \ldots, |V|\}$ and $C = \{\{1, 2\}, \{2, 3\}, \ldots, \{|V| - 1, |V|\}, \{|V|, 1\}\}$. Now if we can find a support of $\mathcal{H}$ in G, this support maps each element in S to a unique vertex in V and each hyperedge in C to a unique edge in E; this support is thus necessarily a Hamiltonian circuit and thus exists if and only if G is a Yes-instance of Planar Hamiltonian Circuit. □

5.4 An Approach Based on Integer Linear Programming

In this section we present our approach using Integer Linear Programming, which is a general method for solving problems of combinatorial optimization. We explained the concept of Integer Linear Programming in Section 3.2 Here, we focus on presenting the *Integer Linear Program (ILP)* for our problem. First, in Section 5.4.1, we describe a basic ILP for finding a hypergraph support of the given set system within the given host graph. We then present extensions to express the preference for compact regions (Section 5.4.2), a relaxed version of the contiguity requirement (Section 5.4.3), and a technique for reducing the number of variables of the ILP (Section 5.4.4).

5.4.1 A Basic Integer Linear Program

We first present the variables, the objective function, and the constraints of an ILP for embedding a set system (i.e., a hypergraph $\mathcal{H} = (S, C)$) into a planar host graph $G = (V, E)$ while ensuring the connectivity of the sets. More precisely, solving the ILP yields a hypergraph support of $\mathcal{H}$ in G if it exists.

Variables. We introduce one binary variable $x_{s,v} \in \{0, 1\}$ for each potential assignment of an element $s \in S$ to a grid vertex $v \in V$ indicating whether it is selected ($x_{s,v} = 1$) or not ($x_{s,v} = 0$).

$$x_{s,v} \in \{0, 1\} \quad \forall (s, v) \in F \tag{5.2}$$

To ensure the contiguity of the sets, we adapt an approach by Shirabe (2009) for contiguity-constrained districting tasks. For this we introduce a directed version $\tilde{G} = (V, \tilde{E})$ of the undirected grid graph $G = (V, E)$, by defining two opposite directed edges for each edge of G, i.e., $\tilde{E} = \{(u, v), (v, u) \mid \{u, v\} \in E\}$. We use

integer variables $y^i_{u,v}$ to model a multicommodity flow in $\tilde{G}$. Index i refers to a set S_i and (u, v) to one directed edge of $\tilde{G}$.

$$y^i_{u,v} \in \{0, 1, \ldots, |S_i| - 1\} \quad \forall i \in \{1, \ldots, k\}, \forall (u, v) \in \tilde{E} \tag{5.3}$$

For each set S_i there is one commodity. We arbitrarily select one element of set S_i as the set's *center*, which we denote as c_i. We will introduce constraints to ensure that the cell to which c_i is assigned serves as the sink of the flow of the commodity for S_i. All other cells to which elements of S_i are assigned serve as sources of that flow. An example is shown in Figure 5.3.

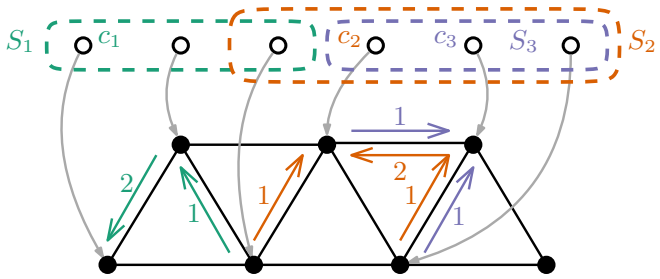


Figure 5.3: An instance and a solution of the basic ILP. Flows of different commodities are displayed with arcs of different colors; they correspond to different sets. The arc labels show amounts of flow. The vertices to which a center (i.e., c_1, c_2, or c_3) is assigned serve as sinks.

Objective. The overall objective is to minimize the sum of costs for selected assignments using the generic cost function $w\colon S \times V \to \mathbb{R}^+_0$.

$$\text{Minimize} \quad \sum_{s \in S} \sum_{v \in V} w(s, v) \cdot x_{s,v} \tag{5.4}$$

Constraints. We first introduce two constraints to ensure that the selected assignments satisfy the definition of an injection.

$$\sum_{v \in V} x_{s,v} = 1 \quad \forall s \in S \tag{5.5}$$

$$\sum_{s \in S} x_{s,v} \leq 1 \quad \forall v \in V \tag{5.6}$$

With the following two constraints we ensure the connectivity of each set based on the multicommodity flow model.

$$\sum_{(v,w) \in \tilde{E}} y^i_{v,w} - \sum_{(u,v) \in \tilde{E}} y^i_{u,v} = \sum_{s \in S_i} x_{s,v} - |S_i| \cdot x_{c_i,v} \quad \forall S_i \in C, \forall v \in V \tag{5.7}$$

$$\sum_{(u,v) \in \tilde{E}} y^i_{u,v} \leq (|S_i| - 1) \cdot \sum_{s \in S_i} x_{s,v} \quad \forall S_i \in C, \forall v \in V \tag{5.8}$$

To demonstrate how the constraints for set S_i control the flow of the corresponding commodity, we discuss three different cases:

(1) If an element of S_i other than the center c_i is assigned to vertex v, the net-outflow from vertex v (left-hand side of Constraint (5.7)) is forced to 1 (right-hand side of Constraint (5.7)). Moreover, the total inflow to vertex v (left-hand side of Constraint (5.8)) is bounded from above by $|S_i| - 1$ (right-hand side of Constraint (5.8)), which is sufficiently large allowing v to receive one unit from every other member of S_i.
(2) If the center c_i is assigned to v, the net-outflow from vertex v is $1 - |S_i|$ according to Constraint (5.7), which means that v receives exactly one unit of flow for every other member of S_i.
(3) If no element of S_i is assigned to v, Constraint (5.8) ensures that v receives no incoming flow. This and Constraint (5.7) together imply that there is no outgoing flow from v.

To summarize, the flow for S_i can only enter or leave vertices to which elements of S_i are assigned. Since every such vertex (except the vertex to which c_i is assigned) injects one unit of flow, the connectivity requirement is satisfied. We note that Constraints (5.7) and (5.8) have been adapted in the following manner from Shirabe's flow model for districting tasks: The sum $\sum_{s \in S_i} x_{s,v}$ on the right-hand sides of the constraints replaces a single variable in Shirabe's model representing whether an area is selected for a certain district.

5.4.2 Compactness of Regions Representing Sets

Setting the cost function w in Objective (5.4) as defined in Equation (5.1), the basic ILP produces a visualization that is compact as a whole. However, this approach does not produce a visualization in which each individual set is represented as a geometrically compact region. To achieve this, we propose two approaches: Measuring compactness based on the regions' perimeters and based on eccentricities.

Compactness based on perimeters. As we discussed in Section 3.3.1, the perimeter of a region is an appropriate measure of its compactness if the region's area is fixed. This holds in our situation since the number of cells for each region is prescribed with the size of the corresponding set and since all cells have the same size. To aggregate the compactness over all regions we simply minimize the sum of their perimeters. We do this by giving a constant bonus for every set S_i and every edge $\{u, v\}$ of G that is within the region for S_i. Maximizing the total bonus indeed corresponds to minimizing the sum of perimeters. This is because every edge of G corresponds to a boundary of constant length between two adjacent cells. Including an edge in a region thus reduces the sum of perimeters by a constant amount.

We use additional binary variables to express the objective:

$$z^i_{u,v} \in \{0,1\} \quad \forall S_i \in C, \forall \{u,v\} \in E \tag{5.9}$$

We force $z^i_{u,v}$ to zero if $\{u,v\}$ is *not* within S_i, i.e., if to u or to v (or to both) no element of S_i is assigned:

$$z^i_{u,v} \leq \sum_{s \in S_i} x_{s,u}\,, \qquad z^i_{u,v} \leq \sum_{s \in S_i} x_{s,v} \qquad \forall S_i \in C, \forall \{u,v\} \in E \tag{5.10}$$

If both u and v are contained in the region for S_i, variable $z^i_{u,v}$ is free to receive value zero or one. Hence, if we include it with a positive coefficient in a maximization objective, it will receive value one whenever possible. Accordingly, with the following objective, we minimize the total length of the boundaries of the regions representing sets:

$$\text{Maximize} \quad \sum_{S_i \in C} \sum_{\{u,v\} \in E} z^i_{u,v} \tag{5.11}$$

In the experiments in which we optimized Objective (5.11) we did not consider any other optimization objective, i.e., we replaced Objective (5.4) with Objective (5.11). However, a combination of the two objectives would be possible by minimizing Objective (5.4) minus Objective (5.11), scaled by some constant factor expressing its importance.

Compactness based on eccentricities. Eccentricity-based compactness metrics usually charge costs proportional to distances or squared distances between a center of a district and centers of the mapping units contained in it. Often, finding appropriate district centers is considered as an integral part of the districting problem at hand (Shirabe, 2009). However, to speed up the computation, it is also common to prescribe the district centers, optimize the assignments of mapping units to the centers, and iterate with a new set of centers computed for the districts (Hess et al., 1965). To implement this approach, we introduce $\mu_1, \ldots, \mu_k \in \mathbb{R}^2$ as the desired geometrical centers of the regions for sets $S_1, \ldots, S_k$. We then apply the basic ILP as presented in Section 5.4.1, but instead of the assignment costs defined in Equation (5.1) we use the following setting:

$$w(s,v) = \sum_{S_i \in C\colon s \in S_i} \|v.p - \mu_i\|^2 \tag{5.12}$$

This means that for every assignment (s,v) we consider the sets containing s. For every such set S_i we charge a cost proportional to the squared distance between the center of the allocated grid cell $v.p$ and the desired geometrical center μ_i of the region representing S_i. Hence, it is favored to assign elements of S_i to cells close

to μ_i. Initially, we set $\mu_1 = \ldots = \mu_k = \mu$, meaning that the center μ of the entire grid is the desired geometrical center for every region. We then compute a solution to the ILP with the assignment costs defined in Equation (5.12). Afterwards, we compute the centers of gravity of the obtained regions and use these as $\mu_1, \ldots, \mu_k$ in a second run of the optimization algorithm. This process is repeated multiple times until the centers $\mu_1, \ldots, \mu_k$ converge. In our examples, the changes of the center of each region become small after the third iteration, and fully converge after five iterations.

When experimenting with the method we made an interesting observation: It took much longer to compute the first solution (with the same center for every region) than the subsequent solutions (with different centers), even if in all iterations we performed a cold start of the solver. A possible reason for this is that with the setting used in the first iteration there are many solutions of exactly the same quality. The solution in Figure 5.2, for example, can be rotated or mirrored without changing the distances between the region centers and the center of the grid. This is not the case, however, with the setting used in the subsequent iterations, where the points $\mu_1, \ldots, \mu_k$ differ from each other. This observation led us to the following approach: In the first iteration, instead of using μ as the center for every region, we draw a very small regular k'-gon centered at μ with k' referring to the number of base map sets. Each of the k'-gon's corners is set as the center of one region of the base map. The center of the overlay sets remains μ. With this approach, the time needed for the first iteration decreased substantially.

5.4.3 Relaxing the Contiguity Requirement

It is easy to imagine set systems that cannot be embedded into a hexagonal or square grid. Think, for example, of seven sets, each of which contains two elements: one element that is shared by all sets and one element that occurs in no other set. Moreover, even if an embedding exists, enforcing contiguity for too many sets can substantially decrease the quality of the solution with respect to the compactness objective.

Therefore, if the task is to visualize many sets as overlays on top of the same base map, we suggest enforcing contiguity only for the regions of the base map. However, the compactness should be optimized for all sets (with respect to any choice of our compactness measures). This approach is always feasible if a sufficiently large grid is used, since the sets of the base map constitute a partition. Our current implementation renders any non-contiguous region as multiple regions, using the same color for their boundaries. For an example, see a visualization of the Austrian parliament in the supplemental material. As a topic for future research, we also consider computing connected visual representations for

non-contiguous regions, e.g., by overlaying linear connections between different parts of a region on top of the boundaries of the base map.

5.4.4 An Integer Linear Program with Fewer Variables

We greatly reduce the number of variables of our ILP by exploiting that, often, two distinct elements $s, t \in S$ occur in exactly the same sets. More formally, we say that $s, t \in S$ are *indistinguishable* if

$$s \in S_i \Leftrightarrow t \in S_i \quad \forall S_i \in C\,. \tag{5.13}$$

Note that when defining the assignment costs according to Equation (5.1) or Equation (5.12), it holds for every two indistinguishable set elements s,t and every cell v that $w(s, v) = w(t, v)$. Hence, given any solution, we can swap the cells of s and t and obtain a solution of the same quality. This means that we can think of s and t as a single element that needs to be assigned to two grid cells. We note that carelessly removing such indistinguishable elements may change the existence of planar support graphs (van Bevern et al., 2016). Hence, we implement this idea by contracting (but not removing) indistinguishable set elements in such a way that we obtain a smaller set system in which each element s represents a number $\alpha(s)$ of original set elements. This smaller set system has the same set relationships as the original one. To assign s to $\alpha(s)$ many cells, we replace the right-hand side of Constraint (5.5) with $\alpha(s)$.

When contracting elements, we need to take care that there remains at least one original element in each set that we can select as the set's center. However, it is not too difficult to contract as many vertices as possible under this requirement.

5.5 Rendering

Our implementation of MosaicSets provides different rendering styles for the base map and set overlays as seen in Figure 5.4. Firstly, we render the base map as a tessellation of either hexagonal or square grid cells. In addition, we add a distance between adjacent cells to better distinguish between the inside and the outside of the overlay regions. As computing the embedding requires a-priori information about the connectivity of the grid graph, the style of the base map must be declared in advance.

Boundary style. Secondly, we provide two different styles to render a set's region as an overlay on the base map. The first style renders the boundary of the region of the union of cells for each set. The boundary itself is drawn on the inner side of the boundary of the occupied cells. Potentially, boundaries of two or more

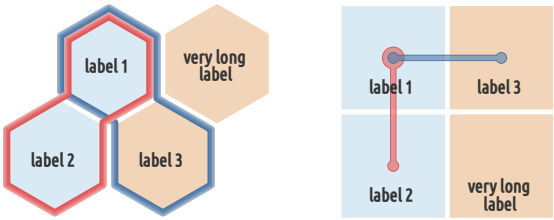


Figure 5.4: Different combinations of proposed rendering styles. The left visualization shows a hexagonal tessellation with boundary style, the right visualization a tessellation with squares and Kelp style.

sets in the same cell could overlap; we define an arbitrary order over all sets, draw the boundaries of sets according to the order but draw the boundary either next to an existing boundary or a cell's boundary. The thickness of the boundary can be specified in advance. To better distinguish different overlay regions, we use a gradient coloring so that the brightness of a region's boundary decreases from the outside to the inside. We refer to this style as *boundary style*.

Kelp style. The second overlay style uses a similar style as seen in Kelp diagrams (Dinkla et al., 2012). As we compute a flow in Section 5.4 we can extract the subgraph of the flow network and use it to draw a Kelp-like overlay. Here, we represent the vertices of the subgraph as filled circles and use thick straight-line segments to represent the edges. Again, we have to define an arbitrary order over the sets first and process sets by said order as otherwise the drawn vertices and edges would overlap if two sets use either the same vertex or edge in the grid graph. By counting the number of sets already using certain edges and vertices in the grid graph, we can scale the diameter of circles and the thickness of straight-line segments such that all sets are visible. We refer to this as *Kelp style*.

Labels. Labels are either placed in the center of a cell or below the center if the Kelp style is used. The label size itself is scaled to fit the label in a cell of the base map. We set a maximum font size and check if all labels fit in their cell. If this is not the case we reduce the font size until either all labels fit or a minimum font size is reached. We automatically apply line breaks for white space and pre-defined delimiter characters, but only render a line break if it is necessary to fit the label in the cell.

Colors. Lastly, the color assignment of the sets is handled by using two palettes of different colors. This decision was guided by the idea that using different colors in each palette allows for a better differentiation between base map and overlays. For the base map we assign colors from a palette of light colors while for the overlays we use bright colors.

5.6 Evaluation

In the following, we assess the quality of MosaicSets by discussing the opinions of experts and evaluating quantitative experiments.

5.6.1 Expert Interviews

To evaluate MosaicSets, we conducted three separate interviews with two domain experts, who are administrative members of the Agricultural Faculty of the University of Bonn, and one designer who works for a design agency in the public transit sector. Prior to the interviews, the two administrative members collaborated to manually create a visualization of the research groups in the Agricultural Faculty of the University of Bonn, which actually inspired the idea of MosaicSets. We note that all visualizations showed in the interviews represent this running example. Combining expertise in the application domain and in graphic design, we consider the three interviewees suitable experts to evaluate our visualizations. The questionnaires with all illustrations and questions can be found in the supplemental material.

Comparison with manually generated visualization. First, we asked the experts to compare the manually generated visualization (see Figure 5.5) with a visualization generated with MosaicSets. Both visualizations show and optimize only the base map. All experts emphasized as the primary difference the intentionally left blank cell in the middle of the manually generated representation where all departments are attached. They explained that the empty cell helps create a common focus and emphasizes the unity of the faculty. However, one expert emphasized that the visualization generated with MosaicSets has less a hierarchical structure with respect to the departments' visualization and seems more flexible in terms of adding new data.

Grid styles. Next, we asked them to comment on one MosaicSets visualization with hexagonal and one with square tiles. They both represented the same sets consisting of six departments and two projects. Although the experts perceived the square tiles as equally clear or slightly clearer than the hexagonal tiles, they emphasized that they generally found the hexagonal style more appealing. They argued that the hexagonal tiles have more linking possibilities, which emphasizes the interrelationship of the departments more strongly. The entirety of the departments appeared as a unit in the hexagonal visualization. As an advantage of the visualization using square tiles, they mentioned the easier countability and identification of individual elements. The arrangement of the elements can easily be translated into rows and columns. Further, the experts pointed out that the

Figure 5.5: Manually generated visualization showing the research groups of the Agricultural Faculty of the University of Bonn. Image credit to Dr. Susanne Plattes and Dipl.-Ing. Michael Kneuper. Figure 5.1 illustrates MosaicSets of the same data set.

visualizations differed with respect to the perceived hierarchy structure. While a hierarchy from the inside to the outside is perceived when hexagonal tiles are used, a hierarchy from the top to the bottom is implied when using square tiles. Overall, the experts saw advantages in both variants. For the running example, the hexagonal grid is more suitable since it better represents the union of the departments. In the remaining part of the interviews we also focused on hexagonal visualizations.

Compactness measures. We asked the experts to compare two versions of MosaicSets generated with different compactness measures. The first version is computed with compactness based on eccentricity. The second is a variant of compactness based on eccentricity that restricts the available grid cells to those used in the first iteration and therefore fixes the overall shape of the grid. In particular, in the first version the focus is on the compactness of each set and in the second version the overall map is more compact. The experts did not show a clear preference. While it is easier to differentiate the individual sets in the first version, the overall appearance is more homogeneous in the second version. The experts agreed that which version to choose depends on the application scenario.

Number of overlays. Comparing MosaicSets with two, three and four overlaying projects, the experts pointed out a problem with visual clutter and separability when there are more than three superimposed projects. Especially, the distinction between inner and outer segments becomes more difficult. Hence, the experts suggested the use of small multiples showing the same base map but different projects.

Interactivity. We also provided the experts with an interactive map allowing the display of only a single project or a chosen selection of projects (see the web page referenced at the end of Section 5.1). When the representation of many projects is important, the experts saw great merits in the interactive map. The experts suggested adding further interactions as highlighting of only the tiles belonging to the selected projects. Further, they proposed adding alternative data representations, for example tables stating cardinality, next to our visualizations. We think these suggestions are well suited to be incorporated into our visualization.

Additional feedback. We also asked the experts for general feedback, and they indicated that the proximity of research groups with joint interests or publications could be another criterion worth considering. Further, they would like to see a consistency criterion that maintains the basic arrangement of the departments' regions when additional institutes, projects, or working groups are added or removed.

Rendering styles. In preliminary interviews we asked two of the experts on their opinion with respect to the rendering. Regarding the overlay style, the experts agreed that boundary style is more suitable than Kelp style. Using Kelp style clarity is lost as the selection of tiles connected with segments is not intuitive. When using boundary style, the experts emphasized that choosing a high-contrast color scheme is important to ensure quick perception of the projects.

Conclusion. Overall, the experts considered MosaicSets to be a valuable approach for visualizing set systems. In particular, the interactive visualization meets the experts' qualitative requirements. They pointed out that there is a high demand for such visualizations in a wide range of application areas, e.g., for internal sessions and meetings as well as for external exposure (e.g., via the faculty website). Due to the highly dynamic structure of the faculty (e.g., research projects starting or expiring) updated visualizations have to be generated frequently. The discussion with the experts revealed that the manual generation of a visualization is highly time-consuming. To create the visualization shown in

Figure 5.5, the corresponding two experts invested several working hours, a large part of which was needed to arrange the tiles. Hence, using MosaicSets can save a substantial amount of valuable working time.

5.6.2 Tasks for Set Visualizations

In the following, we aim to assess MosaicSets with respect to the task taxonomy defined by Alsallakh et al. (2016). The tasks are summarized into three groups: (A) tasks related to elements; (B) tasks related to sets and set relations; and (C) tasks related to element attributes. For some of these tasks we can directly state that MosaicSets is not able to solve them. We cannot solve the tasks of type (C) since we do not consider any element attributes in MosaicSets. Further, tasks A5–A7, B13 and B14 require an interaction technique that MosaicSets does not provide and task B11 asks for set similarity measures that are not defined in our problem setting. To assess which of the remaining 15 tasks can be solved with MosaicSets, we asked the experts from our interviews (see Section 5.6.1) to solve one example for each task and to rate whether a task is 'fully', 'partially', or 'not' supported (see the supplemental material for the full task list and the experts' ratings). We note that the experts' assessment was based on a static visualization with six departments and two projects. All tasks were answered correctly by all experts. The tasks A1, A3, B1–B10, B12 are considered fully supported, and the experts were able to solve them within a few seconds. For tasks A2 and A4 the experts gave different answers; each twice 'fully supported' and once 'partially supported'. The expert who rated the tasks as 'partially supported' took much longer to complete them ($> 30\,\mathrm{s}$).

In summary, we claim that almost all element- and set-based tasks except A5-A7, B11, B13, and B14 are supported by MosaicSets. Compared to other methods from the literature, MosaicSets performs like a combination of Euler-diagrams and frequency grids. In detail, for (A)-tasks MosaicSets performs better than Euler-diagrams, which do not support two (A)-tasks, and it performs nearly as good as frequency grids that support all (A)-tasks. For (B)-tasks MosaicSets performs better than frequency grids but not as good as Euler diagrams. Euler diagrams support all except B13, while frequency grids on the other hand support all (B)-tasks except B3, B4, B11, and B14.

5.6.3 Experimental Setup

We quantitatively compare different versions of MosaicSets: MSP with compactness based on **p**erimeters, MSE with compactness based on **e**ccentricities optimized in multiple iterations, and MSEA, which is a variant of MSE that restricts the used grid cells to the assigned grid cells of the first iteration and

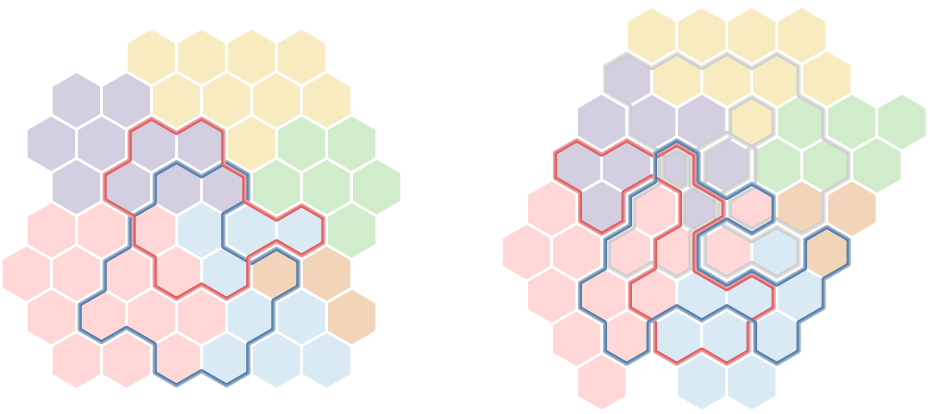
therefore fixes the assigned **a**rea. With all three versions we enforce all sets to form contiguous regions. For all versions we used the elimination of variables presented in Section 5.4.4, as it reduces the running time by an order of magnitude.

We used three data sets: Bonn consists of 51 unique elements and 9 sets, with 6 sets used as base map; Vienna consists of 7 sets with 71 unique elements and 4 sets used as base map; and Parliament consists of 8 sets with 178 unique elements, where 5 sets are used as base map. Bonn refers to the Agricultural Faculty of the University of Bonn and Vienna to the Faculty of Informatics of TU Vienna. Parliament refers to the Austrian parliament with parties as sets of the base map and interest groups as overlays. The host graph has as many rows as columns and is adapted to the number of unique set elements. We add one additional row and column to the minimum required number.

We performed the experiments on an AMD Ryzen 7 PRO 4750U with 16 GB of memory, implemented in Java and used the ILP solver of Gurobi 9.5.1. For all solutions we used a maximum optimality gap in Gurobi of 0.5%. Additional visualizations are provided in the supplemental material.

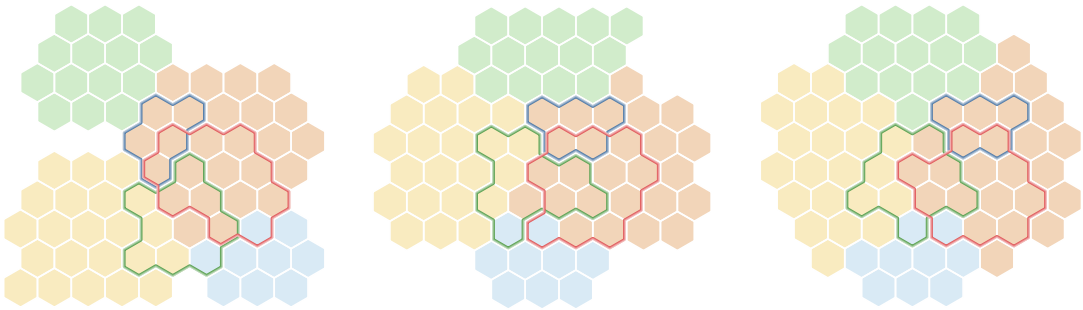
5.6.4 Number of Overlay Sets

From the expert interviews (Section 5.6.1), we learned that overlaying more than three sets (e.g., research projects) severely limits visual clarity. In principle, this problem can be counteracted by using small multiples or interactive maps. For both strategies it is necessary to first generate a visualization containing all sets of interest. Figure 5.6 shows Bonn optimized once with two and once with five overlay sets but both times only two projects are visualized. For an interactive version of Figure 5.6b that allows to display all five overlays see link



(a) two projects (b) five projects

Figure 5.6: MosaicSets computed with eccentricity-based compactness (MSE) for Bonn with different numbers of projects. We highlight the same two projects (red and blue) and indicate the others in gray.



(a) perimeter-based compactness (MSP)

(b) eccentricity-based compactness (MSE)

(c) as (b) but area fixed after first iteration (MSEA)

Figure 5.7: Comparison of the different compactness approaches for VIENNA using three overlay sets

.

in Section 5.1. When applying MSE, the number of overlay sets strongly affects the running time, e.g., from 0.8 s to 40.8 s in Figure 5.6. Computing a solution with four overlay sets took 2.8 s. Moreover, we observe that as the number of considered overlay sets in MSE increases, the compactness of the sets decreases. While the base map in Figure 5.6 is comparably compact in both visualizations, the overlay sets are less compact with five overlays in Figure 5.6b. For the two highlighted projects, the Polsby-Popper score decreases from 0.486 to 0.234 and from 0.500 to 0.236, respectively.

We recommend using small multiples or interactive maps for numerous overlay sets. Otherwise, in case of static maps, using numerous overlay sets results in overloaded and cluttered maps. For the further experiments we focus on static maps and hence limit to three overlay sets which complies with the experts' recommendation.

5.6.5 Running Time

To investigate the influence of the two different compactness formulations (see Section 5.4.2) on the running time, we compare the running times of the variants MSE, MSP and MSEA. We evaluate them for VIENNA with three projects and a hexagonal grid. While both MSE and MSEA were solved in less than one second, we stopped the computation of MSP after one hour with an optimality gap of 15.69%. Thus, we argue that in terms of a reasonable running time, compactness based on eccentricities is a better choice. Nevertheless, we suppose that MSP is well suited to assess the compactness of MSE and MSEA.

Next we focus in more detail on the running times of both versions implementing compactness based on eccentricities: MSE and MSEA. Across all three data sets, the first iteration of MSE needed a maximum of 0.7 s. We observe that the running time of all subsequent iterations is substantially faster. Termi-

nating after a total of five iterations, the combined computation time of the four subsequent iterations is of the same order of magnitude as for the first iteration. Overall, the computation time for a hexagonal grid is $0.7-1.6$ s and for a square grid $0.6-1.6$ s. Considering MSEA, we do not observe a substantial difference in the running time compared to MSE for Bonn and Vienna. However, solving the largest of our data sets, i.e., Parliament, it is about 10% slower. This holds for both the hexagonal and square grid.

5.6.6 Assessing the Compactness

When assessing the compactness of our visualizations, we deem three objectives to be desirable: (C1) the overall base map is compact, (C2) each individual set is compact, and (C3) only base map sets (e.g., departments) are compact. To quantify the compactness, we use the Polsby-Popper score which we denote by PP_{C1} for (C1). For (C2) and (C3) we compute the mean Polsby-Popper score over the considered sets which we denote by PP_{C2} and PP_{C3}, respectively.

In Figure 5.7 we show the results for MSP, MSE, and MSEA for Vienna and a hexagonal grid. We observe that for MSP the overall base map (C1) is less compact compared to the ones achieved by MSE and MSEA which is reflected in $PP_{\text{C1}} = 0.381$, $PP_{\text{C1}} = 0.532$ and $PP_{\text{C1}} = 0.603$, respectively. This can also be observed when considering Figure 5.7, where MSEA results in the most compact overall base map. Considering criterion (C2), we achieve a better result for MSP ($PP_{\text{C2}} = 0.618$) compared to MSE ($PP_{\text{C2}} = 0.574$) and MSEA ($PP_{\text{C2}} = 0.576$). For criterion (C3), we achieve the best result for MSP ($PP_{\text{C3}} = 0.640$). With MSE ($PP_{\text{C3}} = 0.586$) we achieve a slightly lower value regarding (C3). However, fixing the area of the base map after the first iteration with MSEA, we obtain the lowest compactness of these sets ($PP_{\text{C3}} = 0.569$). This reduced compactness was also observed by the experts when they looked at the corresponding visualizations. Overall, we argue that MSE provides a compromise between all criteria while still being applicable in practice considering the running time. With respect to the experts' opinions (see Section 5.6.1), we suggest MSE, since both the compactness of the entire base map (C1) and of the sets building the base map (C3) were considered particularly important.

5.7 Conclusion

We have presented an approach for the visualization of set systems on a regular grid. We require that all sets form contiguous regions and look at two different models for the compactness. In particular, we have a measure based on the perimeter of the region and one based on eccentricities. As the underlying prob-

lem is shown to be NP-hard, we presented ILP formulations of the two variants of the problem.

To evaluate MosaicSets, we interviewed experts about visual criteria and applicability in a real-world scenario. The experts pointed out that MosaicSets is visually as good as a manually generated illustration, while saving several days of labor. Further, we performed experiments with three real-world data sets. In the experiments, we compared the perimeter and eccentricities variants with respect to the compactness; it shows that both have their strengths and weaknesses, but none outplays the other. On the other hand, when comparing the running times it shows that MosaicSets with perimeter compactness runs in more than an hour while MosaicSets with eccentricities compactness can be computed within a few seconds. Hence, we recommend using eccentricities compactness and consider this variant to be applicable in practice.

Still, we see a large potential for future research on MosaicSets. For example, the experts pointed out that combining MosaicSets with other information visualization techniques (e.g., tables) and interaction techniques (e.g., highlighting of cells) can provide a more refined and powerful visualization system. Considering the rendering, we could include the optimization of colors (similar as in GMap (Gansner et al., 2009a,b)) and optimize routing of region boundaries. Further, a comprehensive comparison and support of different rendering styles such as KelpFusion (Meulemans et al., 2013) could give users more flexibility for different use cases. It would be beneficial to develop a user interface for the generation of MosaicSets that includes interactive model manipulation. For example, the user should be able to introduce use-case-specific constraints such as the constraint asking to place each of six departments adjacent to a central empty hexagonal grid cell or specify other global or local shape constraints. Also, dynamically evolving and temporal set systems lead to open research questions, e.g., when considering consistency criteria. Finally, our work brings up interesting new questions for the algorithms community: In which cases (i.e., for which classes of hypergraphs and graphs) can we compute a hypergraph support in a given host graph in polynomial time? Are there efficient approximation algorithms for practically relevant versions? It would be interesting to compare such algorithms with our ILP-based approach.

Chapter 6

Generating Euler Diagrams Through Combinatorial Optimization

The following chapter is based upon a joint publication with Peter Rodgers, Xinyuan Yan, Daniel Archambault, Bei Wang, and Jan-Henrik Haunert (Rottmann et al., 2024). Xinyuan Yan assisted in generating the Euler diagrams using Peter Rodger's visualization approach (Rodgers et al., 2008). The algorithms and experiments were designed and written by myself (Peter Rottmann) and Jan-Henrik Haunert. Peter Rodgers provided comparison data of an existing approach and improved the writing of the comparison. Daniel Archambault provided the source code and assistance in smoothing the Euler diagrams using ImPrEd (Simonetto et al., 2011). All authors discussed the design goals and evaluation of the algorithms. Additionally, all authors provided feedback on the final version of the paper. The authors published another collaborative work on the topic of Euler diagram simplification (Yan et al., 2024) with the focus on set merging.

Can a given set system be drawn as an Euler diagram? Existing methods for deciding this question work only for set systems with special properties. They either split sets into two or more regions, or work on very restricted types of Euler diagrams. We present the first method that correctly decides this question for arbitrary set systems, if the Euler diagram is required to represent each set with a single contiguous region. If the answer is yes, our method constructs an Euler diagram. If the answer is no, our method provides an Euler diagram for a simplified version of the set system where a minimum number of set elements have been removed. Furthermore, we integrate known wellformedness criteria for Euler diagrams as additional optimization objectives into our method. Our focus lies on the computation of a planar graph embedded in the plane to serve as the dual

graph of the Euler diagram. Since even a basic version of this problem is known to be NP-hard, we choose an approach based on Integer Linear Programming (ILP), which allows us to compute optimal solutions with existing mathematical solvers. To this end, we build on previous research on computing planar supports of hypergraphs and adapt existing ILP building blocks for contiguity-constrained spatial unit allocation and the maximum planar subgraph problem. With our exact, ILP-based method we were able to compute optimal solutions for real-world set systems of moderate size, which are typically visualized with Euler diagrams. In addition, we present an efficient heuristic for the generation of Euler diagrams for large set systems, for which the proposed simplification by element removal is indispensable. We report on experiments with data from MovieDB and Twitter. Over all examples, including 850 non-trivial instances, our exact optimization method failed to find a solution for only one set system without removing any set element. However, by removing only a few set elements, the Euler diagrams can be significantly improved with respect to our well-formedness criteria.

6.1 Introduction

Euler diagrams are frequently used to visualize set systems. They represent each set as a region in the plane that is bounded by a closed curve. An area in an Euler diagram that is occupied by one or multiple regions indicates the existence of set elements that are contained in the corresponding sets and in no other set. An advantage of Euler diagrams is that they are intuitive to understand. However, they can become cluttered even for medium-sized set systems. For a given set system, it is possible that no Euler diagram exists when requiring the regions to be contiguous. Previous approaches (Rodgers et al., 2008; Simonetto and Auber, 2008) split sets into two or more separate regions. However, splitting makes it harder to identify regions belonging to the same set, and increases the overall number of regions in the visualization.

Our work addresses two research problems. First, none of the existing methods can decide for a given set system whether an Euler diagram with a single contiguous region for each set exists, let alone generate an Euler diagram in every case where it is possible. Second, Euler diagrams have typically been considered as a tool to visualize set systems without loss of information. We think Euler diagrams can be used as a tool to visually summarize large set systems, but for this summary, the generation of Euler diagrams has to be combined with a simplification of the given data.

Contribution. Our contribution is a new combinatorial optimization approach. Given a set system as input, our goal is to compute a simplified but still similar

version of the set system that can be drawn nicely as an Euler diagram. The simplification is achieved by removing some set elements from the set system. To quantify the loss of information caused by the removal of elements, we assume that every element has a weight expressing the cost of its removal. For example, the weight of each element may be one, or it can be equal to the number of sets containing the element. More generally, any measure expressing the importance of the element can be applied.

Our most important contribution is an exact optimization method that minimizes the total weight of the removed set elements while ensuring that the resulting simplified set system can be embedded in the plane as an Euler diagram with a single contiguous region for each set. Moreover, we contribute multiple extensions of our method to integrate additional optimization criteria that improve the resulting Euler diagram with respect to known wellformedness conditions (Rodgers et al., 2011). For large set systems, we introduce an efficient heuristic method. We conduct experiments with both the exact method and the heuristic on realistic set systems and assess the quality of their solutions as well as their efficiency.

Outline. In the following, we first define a basic version of the problem; see Section 6.2. Then, in Section 6.3, we give an overview of the complete workflow from a set system to an Euler diagram. In Section 6.4, we present our exact and heuristic optimization methods. We evaluate both approaches in Section 6.5 and conclude this Chapter in Section 6.6.

6.2 Preliminaries and Basic Problem

In order to construct an Euler diagram for a given set system, it is sufficient to find the dual graph of the Euler diagram which should be a planar graph in order to produce wellformed Euler diagrams; see Section 2.2. The dual graph corresponds to a planar support of the hypergraph $H = (V, E)$ representing the set system, where V is the set of set elements and E is the set of sets; see Section 3.1.2. Removing set elements from a set system to simplify the set system is equivalent to removing a vertex from the hypergraph. When we remove twins in the hypergraph, multiple set elements are represented by a single vertex of the hypergraph. As a consequence, the vertex is removed only if all elements corresponding to that vertex are also removed.

When we say that we remove an element v from a hypergraph $H = (V, E)$, we imply that v is removed from the vertex set V of H as well as from every hyperedge $X \in E$ in which v occurs. We introduce for every vertex $v \in V$ a weight $w(v) \in \mathbb{R}_{>0}$ measuring the cost of its removal. If the aim is to minimize

the number of element removals, we set $w(v) = 1$ for each set element. Else, if the aim is to minimize the loss of set memberships, we set $w(v) = |\ell(v)|$, where $\ell(v)$ is the set of hyperedges containing v, i.e., $\ell(v) = \{X \in E \mid v \in X\}$. With $H - S$, we refer to the hypergraph resulting from a hypergraph $H = (V, E)$ after removing a set $S \subseteq V$ from it. With this, we are ready to state a basic problem that asks for an optimal simplification of a set system to generate an Euler diagram.

Problem 3 (SetSystemSimplification)**.** *Given a hypergraph $H = (V, E)$ and a weighting $w\colon V \to \mathbb{R}_{>0}$, find a minimum-weight set $S \subseteq V$ such that $H - S$ has a planar support and return such a planar support as output.*

We would like to point out that SetSystemSimplification has a solution with $S = \emptyset$ if and only if the given hypergraph H has a planar support. Therefore, any exact optimization algorithm for SetSystemSimplification can be used as a tool to decide whether a hypergraph has a planar support. Since deciding whether a hypergraph has a planar support is NP-hard (Johnson and Pollak, 1987), we conclude that SetSystemSimplification is NP-hard, too.

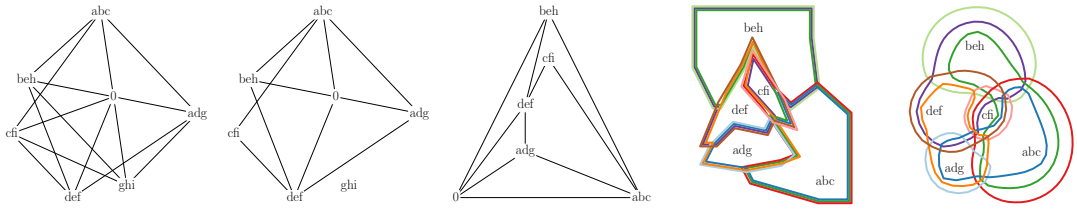
As our problem is NP-hard, there is no reasonable hope for an efficient exact algorithm. Therefore, we focus on an approach based on integer linear programming, which is a general method for solving combinatorial optimization problems and has the advantage that we can employ existing mathematical solvers. With this, we can solve both the new problem SetSystemSimplification and the known problem of finding a planar support for a given hypergraph.

Solving SetSystemSimplification is insufficient since it neglects important criteria. Therefore, we will extend it to a multi-criterial problem, MCSetSystemSimplification, which is based on the wellformedness conditions for Euler diagrams presented in Section 2.2. Both our exact method and our heuristic can deal with this extended problem.

6.3 Workflow

While we consider the exact optimization method and the heuristic for MCSetSystemSimplification as our main contribution, we also provide a complete workflow that yields an Euler diagram visualizing a simplified version of a given hypergraph. This workflow is illustrated in Figure 6.1 for an artificial example and outlined below.

Step 1: Condensed hypergraph. In Step 1 of our workflow, we replace every set of vertices that are contained in exactly the same hyperedges with a single vertex, yielding the *condensed hypergraph*. The motivation for this step is that vertices contained in exactly the same hyperedges should not be separated in the



(a) superdual graph (b) dual graph (c) planar embedding (d) Euler diagram (e) resulting diagram

Figure 6.1: Results after different steps of our hypergraph visualization workflow. The input to the shown example is a set system consisting of nine sets, i.e., hyperedges: `a`: $\{1,4,7\}$, `b`: $\{1,5,7\}$, `c`: $\{1,6,7\}$, `d`: $\{2,4\}$, `e`: $\{2,5\}$, `f`: $\{2,6\}$, `g`: $\{3,4\}$, `h`: $\{3,5\}$, `i`: $\{3,6\}$. The set elements 1 and 7 are replaced by a single set element in the first step because both are contained by the same hyperedges. The label of each vertex refers to the sets containing it. The additional vertex labeled 0 represents the outer face. The results of Steps 2–6 are visualized by Figs. (a)–(e). For each set, there exists one edge in the superdual graph that has to be selected to ensure the set's connectivity if all set elements are selected. Since these edges constitute the non-planar graph $K_{3,3}$, a vertex of the superdual graph has to be removed to generate a planar dual graph.

final visualization. Moreover, working with the condensed graph greatly reduces the running time of our methods. However, this step may affect the existence of a planar support (van Bevern et al., 2024). Therefore, we may optionally skip this step, for example, if the aim is to compute a planar support of the original hypergraph.

Step 2: Superdual graph. In Step 2 of the workflow, the condensed hypergraph is used to compute the *superdual graph*, whose vertex set contains every vertex of the condensed hypergraph as well as a special vertex v_0 representing the outer face, and whose edge set contains every edge that may be useful for the dual graph of the Euler diagram. Every edge which shares a label between two vertices in the condensed hypergraph can potentially connect two vertices within the same set.

Step 3: Dual graph. Next, in Step 3, a solution to MCSetSystemSimplification is computed in the form of a selection of vertices and edges of the superdual graph, which determines both the simplified hypergraph and the dual graph of the Euler diagram. This is achieved either with our exact optimization approach via integer linear programming or our heuristic method. This step of the processing pipeline is our main focus of the work and is discussed in detail.

Step 4-6: Euler diagram generation. We use existing methods for the last three steps of our workflow. First, in Step 4, we compute a planar embedding of the dual graph generated in Step 3 (Chrobak and Payne, 1995). In Step 5, the planar embedding is used to construct an initial Euler diagram (Rodgers et al.,

2008) that is guaranteed to contain only simple curves. Finally, in Step 6, the curves of the initial Euler diagram are smoothed (Simonetto et al., 2016).

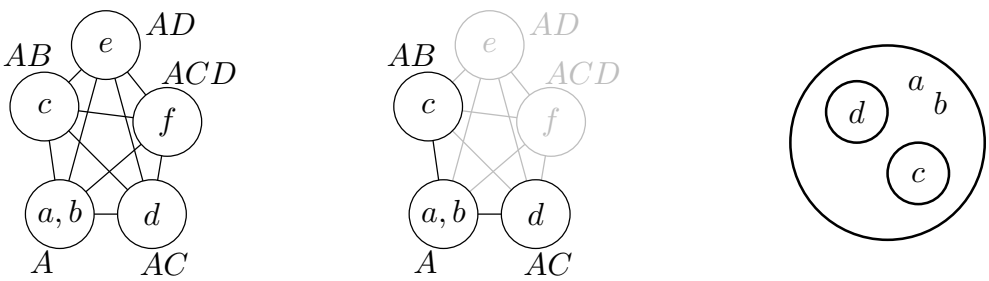
The properties of the resulting Euler diagram are as follows:

- Each set forms a single, contiguous region
- Maximized set memberships
- Minimal concurrency

6.4 Methodology

In this section, we present our exact method and our heuristic for MCSetSystemSimplification; see Section 6.4.1 and Section 6.4.2, respectively. Recall that both methods take the superdual graph $G = (Z, F)$ as input and return a planar subgraph $G' = (Z', F')$ of it, which will serve as the Euler diagram's dual graph.

The prerequisite for our algorithm is the creation of the superdual graph $G = (Z, F)$ with zones as vertices and undirected edges F of the given set system. These edges indicate shared labels between the incident zones. Hence, an edge is added between pairs of zones which share at least one label; see Figure 6.2a.



(a) superdual graph (b) dual graph (c) Euler diagram

Figure 6.2: From left to right: (a) the superdual graph G for S, (b) the subgraph G' of G that has been computed to form the dual graph of the Euler diagram, (c) the Euler diagram.

For the case of Euler diagrams, an edge in the dual graph of the Euler diagram indicates that those zones will share a border in the drawn diagram; see Figures 6.2b and 6.2c. The number of label differences indicates the number of contour lines which will be crossed at the shared border. Every additional contour line is resulting in an increased concurrency. We count the label differences and set them as edge weights. Hence, the weight of an edge $w(u, v)$ is defined by the symmetric difference Δ of labels of the adjacent zones u and v minus one.

$$w(u, v) = |\ell(u)|\Delta|\ell(v)| - 1 \tag{6.1}$$

Our goal is to retrieve a planar subgraph $G' \subseteq G$ which is the dual graph of the Euler diagram. This graph should maximize the drawn zones and minimize the overall concurrency of the drawing. In doing so, each induced subgraph of a

single label needs to form a connected component. Zones, which introduce non-planarity to the subgraph while enforcing the contiguity of sets, are not part of the planar subgraph.

6.4.1 An Integer Linear Program

Variables encoding the selected subgraph. For each vertex $u \in Z$ of the superdual graph $G = (Z, F)$, we define a binary variable x_u indicating whether the vertex is part of the selected subgraph G' ($x_u = 1$).

$$x_u \in \{0, 1\} \quad \forall u \in Z \tag{6.2}$$

Moreover, we introduce one binary variable $z_{u,v}$ for each edge $e = \{u, v\} \in F$. This variable indicates whether e is selected for G'.

$$z_{u,v} \in \{0, 1\} \quad \forall \{u, v\} \in F \tag{6.3}$$

These z-variables are coupled with the x-variables to ensure that an edge can be selected only if its two incident vertices are selected.

$$z_{u,v} \leq x_u \quad \text{and} \quad z_{u,v} \leq x_v \quad \forall \{u, v\} \in F \tag{6.4}$$

Objective. The x- and z-variables introduced in the previous paragraph are sufficient to express all three objectives of MCSetSystemSimplification.

$$\text{Maximize} \quad \sum_{p \in Z} w(p) \cdot x_p - \alpha \sum_{\{u,v\} \in F_1} \omega(u, v) \cdot z_{u,v} + \beta \sum_{\{v_0,v\} \in F_2} z_{v_0,v} \tag{6.5}$$

Enforcing connectivity of hyperedges. To ensure the connectivity of hyperedges in the selected subgraph G' of G, we adapt a flow model by Shirabe (2005) for spatial unit allocation tasks occurring in spatial planning. In our application, we apply the model separately to each hyperedge $X \in E$. For this, we introduce a directed graph $\tilde{G}_X = (Z_X, A_X)$; see Figure 6.3. The vertex set of $\tilde{G}_X$ contains every vertex of G that is labeled with X, i.e.,

$$Z_X = \{u \in Z \mid X \in \ell(u)\}. \tag{6.6}$$

The edge set of $\tilde{G}_X$ contains two opposite, directed edges for each edge of G that connects two vertices labeled with X, i.e.,

$$A_X = \{(u, v), (v, u) \mid \{u, v\} \in F, X \in \ell(u), X \in \ell(v)\}. \tag{6.7}$$

Due to the definition of G, $\tilde{G}_X$ is a complete, directed graph. Additionally, the vertices of $\tilde{G}_X$ share at least one label.

Figure 6.4 illustrates the flow model that ensures the connectivity of the subset $\{u \in X \mid x_u = 1\}$ selected from X. Each vertex of $\tilde{G}_X$ except the sink vertex can be the source of flow. Hence, the maximum flow that can be transferred across an edge (u, v) is limited to the number of vertices in X minus one, i.e., $|X| - 1$. We represent the flow in $\tilde{G}_X$ with the following variables:

$$y^X_{u,v} \in \{0, \ldots, |X| - 1\} \quad \forall X \in E, \forall (u, v) \in A_X. \tag{6.8}$$

These variables are constrained such that only directed edges corresponding to edges of G that are selected for G' can carry flow.

$$y^X_{u,v} \leq (|X| - 1) \cdot z_{u,v} \quad \forall (u, v) \in A_X, \forall X \in E \tag{6.9}$$

Another set of variables models which vertex acts as a sink of the flow network for X.

$$c^X_u \in \{0, 1\} \quad \forall X \in E, \forall u \in Z_X \tag{6.10}$$

These variables are constrained such that the network for X can contain at most one sink (vertex e in Figure 6.4, with $c^X_e = 1$).

$$\sum_{u \in Z_X} c^X_u \leq 1 \quad \forall X \in E \tag{6.11}$$

Finally, the next two constraints ensure that

- every selected vertex except the sink contributes at least one unit of flow to the network (i.e., vertices b, c, d in Figure 6.4),
- the sink (i.e., vertex e in Figure 6.4) is allowed to receive as much flow as there are vertices in the network, and
- every non-selected vertex receives no flow (i.e., vertex a in Figure 6.4).

$$\sum_{(v,w) \in A_X} y^X_{v,w} - \sum_{(u,v) \in A_X} y^X_{u,v} \geq x_v - |X| \cdot c^X_v \quad \forall X \in E, \forall v \in Z_X \tag{6.12}$$

$$\sum_{(u,v) \in A_X} y^X_{u,v} \leq (|X| - 1) \cdot x_v \quad \forall X \in E, \forall v \in Z_X \tag{6.13}$$

To summarize, every flow that a vertex contributes to the network for a hyperedge X has to reach the sink and can pass only through vertices selected for X. The presented model up to now is sufficient to ensure that the vertices selected for a hyperedge X induce a connected subgraph in the selected subgraph G' of G.

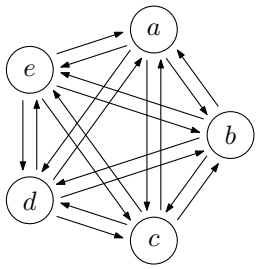


Figure 6.3: The directed graph $\tilde{G}_X$ for a hyperedge X.

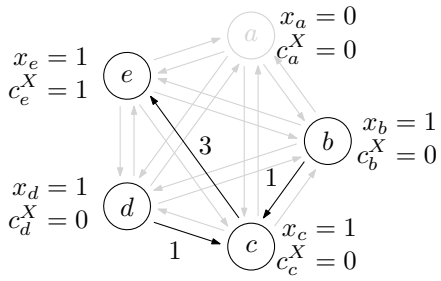


Figure 6.4: A feasible assignment of the variables modeling the connectivity of the selected subgraph for X.

Enforcing planarity. A graph is planar if and only if it does not contain a Kuratowski subdivision as a subgraph (i.e., a subdivision of $K_{3,3}$ or K_5) (Kuratowski, 1930). Hence, we can ensure the planarity of the output graph G' by requiring that, for each Kuratowski subdivision K contained in G, at least one edge is unselected. For this, we add the constraint

$$\sum_{\{u,v\} \in F(K)} z_{u,v} \leq |F(K)| - 1 \quad \forall K \in \mathcal{K}(G)\,, \tag{6.14}$$

where $\mathcal{K}(G)$ is the set of all Kuratowski subdivisions contained in G, and $F(K)$ is the set of edges of a graph K.

A challenge with this formulation is that there can be exponentially many Kuratowski subdivisions in G. Therefore, setting up the ILP with all $|\mathcal{K}(G)|$ instances of Constraint (6.14) is prohibitive in practice. To tackle this challenge, we initially set up the ILP without Constraint (6.14), but we make sure that the solver detects relevant instances of Constraint (6.14) during the optimization process and adds them to the model. We implement this idea with a *callback*, which is a customized method that the solver automatically invokes at certain states of the optimization process. The callback that we introduce is invoked whenever the solver finds a new *incumbent solution*, i.e., a solution that satisfies all constraints of the current model and that is better than any solution found before. For the incumbent solution that led to the invocation of the callback, let G' be the subgraph of G as determined with the z-variables. We apply the Boyer-Myrvold's planarity testing algorithm (Boyer and Myrvold, 2004) to G', which either reports that G' is planar or returns a Kuratowski subdivision K of G', thus proving that G' is non-planar. If G' is planar, we simply resume solving the current model, since there may be better solutions than G' that the solver has not found yet. Else, we instantiate Constraint (6.14) with the found Kuratowski subdivision K and add this instance of the constraint to the model, before resuming the solution procedure. This strategy is supported by state-of-the-art ILP solvers in a way that guarantees an optimal solution as output.

To strengthen our initial model without Constraint (6.14), we add the following constraint if G has at least 3 vertices.

$$\sum_{\{u,v\} \in F} z_{u,v} \leq 3 \cdot \sum_{p \in Z} x_p - 6 \tag{6.15}$$

This inequality does not ensure planarity but, due to Euler's formula, is valid for all solutions satisfying the planarity requirement.

6.4.2 Heuristic Approach

We now present our heuristic for selecting a planar subgraph $G' = (Z', F')$ of the superdual graph $G = (Z, F)$. The heuristic uses the same superdual graph as the

exact, ILP-based method. However, the heuristic does not need to create directed subgraphs $\tilde{G}_X$ for each individual hyperedge X. The heuristic initializes G' with the vertex representing the outer face as the only vertex, i.e., $Z' = \{v_0\}$ and $F' = \emptyset$. Adding v_0 to G' ensures that the outer face is always part of the solution which is necessary to produce a valid dual graph. Then, it lets G' grow in an iterative and greedy manner, ensuring that after every iteration the connectivity requirement for hyperedges and the planarity requirement are satisfied. Thus, the heuristic guarantees a feasible solution to MCSetSystemSimplification, but not an optimal solution.

Algorithm 2 describes the heuristic in more detail. The iterative growth is implemented with a while loop, where in each iteration the method ADDNEXTVERTEX is called. The method returns true or false, depending on whether it succeeded to grow G'. If the method was unsuccessful, there is no vertex $v \in Z \setminus Z'$ that can be added to G' without violating any constraint. Thus, the algorithm terminates and returns G'.

Algorithm 2 Heuristic set system simplification

Input: Undirected superdual graph $G = (Z, F)$ with vertex weights w and edge weights ω

```
procedure SIMPLIFY(G)
    Z' ← {v_0}, F' ← ∅, G' ← (Z', F')
    s ← True
    while s do
        s ← ADDNEXTVERTEX(G', G)
    end while
    return G' = (Z', F')
end procedure
```

The method ADDNEXTVERTEX is presented in Algorithm 3. It first computes a list C of *candidates*, where a candidate c is an extension of G' by one vertex $c.v \in Z \setminus Z'$ and a set of edges $c.edges$ between $c.v$ and vertices of G'. The candidate list C is obtained as the union of multiple lists, each of which is computed based on one edge $\{p, v\} \in F$ with $p \in Z'$ and $v \in Z \setminus Z'$, using a method CREATEEDGECANDIDATES. This method returns only candidates whose selection satisfies the connectivity requirement (as we will later explain). After C has been set up, we compute for each candidate $c \in C$ the increase in the objective value that its selection would cause and store it as $c.w$. Finally, we go through the candidates in decreasing order of $c.w$ and choose the first candidate that does not violate the planarity requirement. For planarity testing, we use the Boyer-Myrvold algorithm (Boyer and Myrvold, 2004), as in our ILP-based method.

The method CREATEEDGECANDIDATES yields for a given edge $e = \{p, v\}$ a set of candidates; see Algorithm 4. For every candidate c in this set, $c.v = v$ and $c.edges$ contains e. However, the algorithm returns multiple candidates, which differ with respect to the edges in $c.edges$ in addition to e. To compute this set, let $X_1, \ldots, X_K$ be the hyperedges that contain v and whose connectivity would not be achieved when selecting v with edge $e = \{p, v\}$ alone. For example, in Figure 6.5, this holds for the hyperedge `a`. For each such hyperedge, the connectivity can be repaired with a single edge among the edges incident to v other than e, whose number is $\deg(v) - 1$, where $\deg(v)$ is the degree of v in G'. The addition of a single hyperedge can also repair the connectivity of multiple hyperedges. Using such edges reduces the number of edges of a candidate but

Algorithm 3 Greedy addition of the next vertex together with edges connecting it to the current graph

Input: Current selected dual graph $G' = (Z', F')$, superdual graph $G = (Z, F)$

```
1:  procedure ADDNEXTVERTEX(G', G)
2:      ▷ Collect candidates:
3:      C ← empty list of candidates
4:      for (p, v) ∈ F such that p ∈ Z' and v ∉ Z' do
5:          C_pv ← CREATEEDGECANDIDATES(G', G, p, v)
6:          C ← C ∪ C_pv
7:      end for
8:      ▷ Rate candidates:
9:      for c ∈ C do
10:         c.w ← w(c.v) − α · Σ_{e∈c.edges∩F_1} ω(e) + β · |c.edges ∩ F_2|
11:     end for
12:     ▷ Choose the best candidate:
13:     Sort C descending by weight w
14:     while C.size > 0 do
15:         c ← C.pollFirst()
16:         Add vertex c.v to Z' and edges c.edges to F'
17:         if isPlanar(G') then
18:             return True
19:         else
20:             Remove c.v from Z' and c.edges from F'
21:         end if
22:     end while
23:     return False
24: end procedure
```

Algorithm 4 Create candidates for a single edge $\{p, v\}$

```
1: procedure CREATEEDGECANDIDATES(G', G, p, v)
2:     C ← empty list of candidates
3:     Let {X_1, ..., X_K} be the set of hyperedges in ℓ(v) whose connectivity
       would be violated if selecting only v and {p, v}.
4:     For i = 1, ..., K let F'_i be the set of edges connecting v with a selected
       vertex of hyperedge X_i, i.e., a vertex in X_i and G'.
5:     for ϵ ∈ F'_1 × F'_2 × ... × F'_K do
6:         Let c be a new candidate
7:         c.v = v
8:         c.edges = set containing edge {p, v} and all edges in ϵ
9:         C.add(c)
10:    end for
11:    return C
12: end procedure
```

increases the number of candidates due to additional combinations. However, we must consider both options: adding multiple edges is more likely to produce non-planar solutions; adding a single edge that repairs multiple hyperedges can lead to higher concurrency.

Hence, we generate $O((\deg(v) - 1)^K)$ candidates, where $K < |\ell(v)|$. In Figure 6.5, this yields two candidates, visualized with different colors. Explicitly enumerating the candidates can be done reasonably fast, assuming that the number $|\ell(v)|$ of hyperedges containing vertex v is a small constant, which we did in all our experiments with the heuristic. For set systems where $|\ell(v)|$ can be large, one may use our heuristic with a user-set upper limit on the size of *c.edges*. However, restricting the candidate space by removing all candidates with a high edge count can degrade the quality of the solution.

Figure 6.5: Graph G' (black) after adding three vertices and three edges with our heuristic. Vertex `ab` has not been added yet (left). Calling CREATEEDGECANDIDATES for edge {`a`, `ab`} yields two candidates: The red candidate with edges {`a`, `ab`},{`b`, `ab`} and the blue candidate {`ac`, `ab`},{`b`, `ab`} (right).

6.5 Experiments

Datasets. We evaluate our methods with two datasets.

- MovieDB from the 2007 InfoVis contest (Kosara et al., 2007): Each set system is based on a single director. The sets are the movies of the director. The set elements are actors of the corresponding movie.
- TwitterCircles (Leskovec and Krevl, 2014): The dataset consists of users and their interest groups. Each set system is based on a user's ego network. Each set is based on the user's interest circles while the set elements are followed users belonging to at least one interest circle.

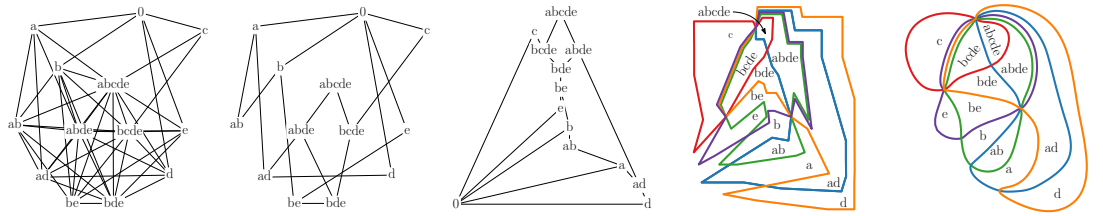
In total, there are 930 set systems in TwitterCircles and 16884 in MovieDB. However, we omit all set systems that have less than five vertices in addition to v_0 in the superdual graph since those always have a planar embedding, simply because they cannot have a K_5 or a $K_{3,3}$ as a minor. Including a lot of trivial instances in the experiments would decrease the differences between our two approaches. Thus, we select 281 and 569 set systems, respectively. Table 6.1 provides further details. For all experiments, we defined the weight of an element as the number of sets containing it, and we applied the optional Step 1, i.e., the condensation of the hypergraph. All experiments were executed on an AMD Ryzen 9 7950X with 64 GB of RAM using Java 11 and Gurobi 9.5.1 for solving the ILPs.

The code and datasets are available under a GPL open source license from `https://gitlab.igg.uni-bonn.de/geoinfo/generating-euler-diagrams`.

Before analyzing the results, we show the full workflow on the set system for the director Keith Hooker from MovieDB in Figure 6.6. The set system consists of 5 sets with 19 set elements. Our ILP approach produces an Euler diagram where each single-labeled face is adjacent to the outer face. Additionally, the minimization of the number of concurrencies results in a nested structure. Vertices with only a few labels are placed near the outer face. In contrast, regions with a higher label count are moved toward the center of the Euler diagram.

dataset	#set systems		#sets	#elements
MovieDB	569	Avg	4.39	44.30
		Max	15	288
TwitterCircles	281	Avg	6.19	73.83
		Max	14	197

Table 6.1: Statistics of MovieDB and TwitterCircles.



(a) superdual graph (b) planar support (c) planar embedding (d) Euler diagram (e) resulting diagram

Figure 6.6: Euler diagram workflow of the set system corresponding to the director Keith Hooker from MovieDB. The set system consists of 5 sets with 19 set elements. Condensing the hypergraph in Step 1 yields 12 unique set elements. The results of Steps 2–6 are visualized by Figs (a)–(e). When visualizing this set system with $\alpha = 0.01$ and $\beta = 0.1$, no set element is removed from the set system.

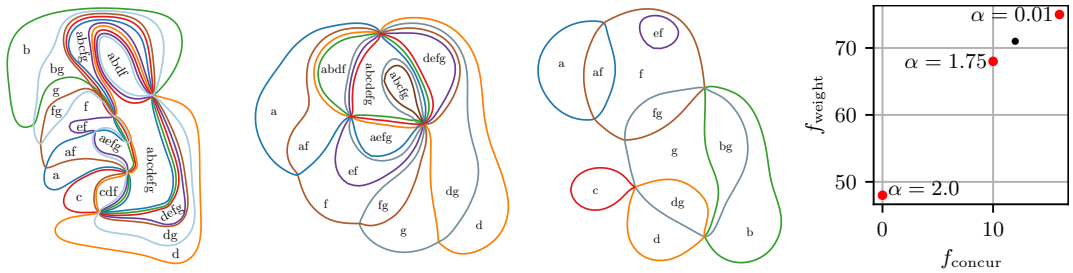


Figure 6.7: Simplifying the set system of director Joel Schoenbach from left to right. The left figure is computed using $\alpha = 0.01$ with $f_{\text{weight}} = 75$ and $f_{\text{concur}} = 16$. The middle figure yields $f_{\text{weight}} = 68$ and $f_{\text{concur}} = 10$ for $\alpha = 1.75$. The lowest concurrency $f_{\text{weight}} = 48$ and $f_{\text{concur}} = 0$ is a result of $\alpha \geq 2.0$. $\beta = 0.1$ is fixed for all results. The diagram on the right shows the relation between f_{weight} and f_{concur} for the shown example when changing α. The red data points correspond to the visualized Euler diagrams.

Parameter influence. To study the influence of the parameters α and β, we computed multiple solutions for all set systems from MovieDB using our ILP approach. For $\beta = 0.1$ and $\alpha = 0.01$ the average number of concurrencies was 1.43 and all set elements were selected. Keeping β fixed and increasing α to 1.25 reduces the average number of concurrencies per Euler diagram by 54.98%, whereas the weight of the selected set elements decreases by 0.42%.

Our goal is to demonstrate the complexity of the datasets and the advantages of our simplification through a single example. Therefore, the director Joel Schoenbach is shown in Figure 6.7. This example shows that our approach can simplify the set system to reduce concurrency. However, this comes at the cost of the removal of elements. We show additional examples of the influence of α and β in the supplementary material.

Similarly, we investigate the influence of the parameter β. We compute two solutions for the same set system; see Figure 6.8. Both solutions were obtained with $\alpha = 0.01$. On the left, we set $\beta = 0.1$. On the right, we set $\beta = 0$, which results in only a single vertex being adjacent to the outer face. The latter leads to

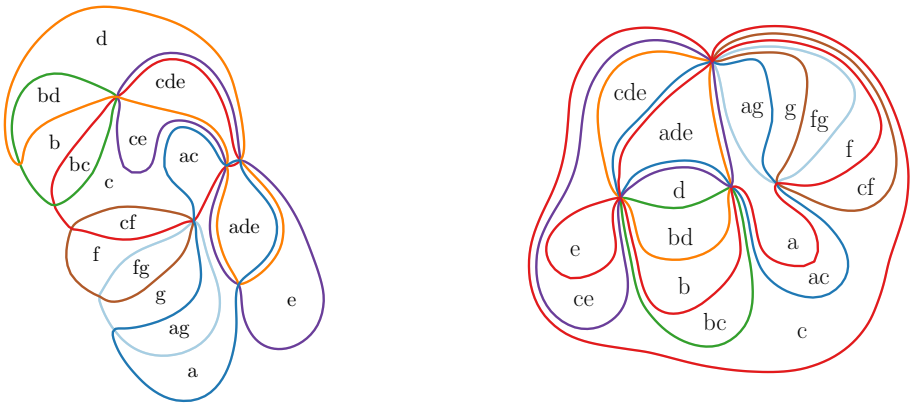


Figure 6.8: Comparison of results for director Art Camacho for $\beta = 0.1$ (left) and $\beta = 0$ (right); $\alpha = 0.01$ for both. For $\beta = 0$, only one vertex of the dual graph, i.e. labelled c, is adjacent to the outer face.

a hardly readable visualization. The sets are nested and form holes within the set that is connected to the outer face. Comparing both Euler diagrams, it is harder to follow individual set outlines for $\beta = 0$. As a result, it would be difficult to determine labels within the interior of the diagram when only a single label for each curve is given.

Comparison of solutions of exact method and heuristic. We now compare the results of the ILP and of the heuristic for the 850 set systems given with the two datasets. For this purpose, we evaluate the value of the overall objective function f and the functions f_{weigth}, f_{concur}, f_{outer} expressing the three criteria of MCSetSystemSimplification. Based on our experiences with the different parameter settings discussed in the previous section, we fixed $\alpha = 0.01$ and $\beta = 0.1$ for all the experiments that we report here.

We use a time limit of one hour for the ILP solver and report the best solution. For all 850 set systems, the solver found a solution within the time limit, but in four examples from TwitterCircles the solver was not able to prove the optimality of the solution. In three cases, the solution contained 100% of the total weight of the set system. In one example (id = 779715), a solution with 50.82% of the total weight was returned and a gap of 96.64% was reported, meaning that there might be a solution whose objective value is roughly twice as high as that of the found solution.

The goal of the first experiment is to compare the quality of our heuristic and our ILP. For doing this, we solve all instances of the two datasets with both approaches. Recall that we aim at high values for f, f_{weight} and f_{outer} but low values for f_{concur}. Across all instances, we observe that the heuristic performs worse than the ILP, in the sense that it achieves a lower value for the objective function f (see the column f in Table 6.2 for MovieDB and Table 6.3

Method		f	f_{weight}	f_{concur}	f_{outer}	Time [s]
ILP	Avg	50.68	50.26	1.43	4.08	0.221
	Max	294.66	294	55	14	99.3485
Heuristic	Avg	50.68	50.26	2.78	4.07	0.0003
	Max	294.66	294	90	14	0.018

Table 6.2: Results of the ILP and the heuristic on MovieDB. Avg and Max are the average and maximum over all instances.

for TwitterCircles). This is mainly due to two factors: the heuristic includes less vertex weight in the dual graph (column f_{weight}) and selects edges with more concurrency (column f_{concur}).

For all instances in MovieDB, our ILP approach found a planar support with all set elements. Hence, the metric f_{weight} is the same for the input data and the results. In contrast, the heuristic did not include all set elements in every solution. The average value of f_{weight} is 0.05% less with the heuristic than with the ILP. For the instances in TwitterCircles, the ILP in a single case yielded a solution without all set elements, and the average value of f_{weight} is 0.16% less with the heuristic than with the ILP.

To evaluate the results with respect to concurrent curves, we compare the average values of f_{concur} of our approaches. For TwitterCircles, f_{concur} increased from 2.02 to 4.12. For MovieDB, f_{concur} increased from 1.43 to 2.78. The increased number of concurrencies is also reflected by the maximum values of f_{concur}, which increased from 89 to 111 and from 55 to 90, respectively.

We investigate the distribution of the increased number of concurrencies by the factor of two in Figure 6.9. None of the instances drastically exceeds this factor. The additional concurrencies are well distributed across all instances in MovieDB. As an example, the results of both approaches for the director Brett Ryan Bonowicz are shown in Figure 6.10. In that case, the additional concurrency is well distributed across the visualization, and the maximum number

Method		f	f_{weight}	f_{concur}	f_{outer}	Time [s]
ILP	Avg	128.73	128.20	2.02	5.06	54.05
	Max	1069.16	1069	89	13	3,602.53
Heuristic	Avg	128.50	127.99	4.12	5.00	0.003
	Max	1069.08	1069	111	13	0.267

Table 6.3: Results of the ILP and the heuristic on TwitterCircles. Avg and Max are the average and maximum over all instances.

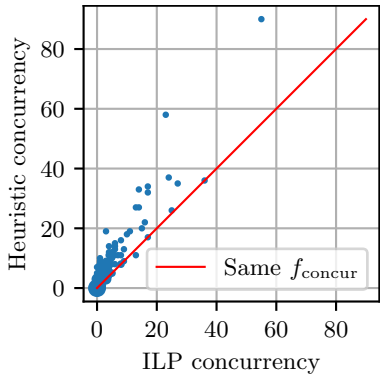


Figure 6.9: Comparison of concurrency of our heuristic and ILP approach. Each point represents a set system in MovieDB.

of concurrencies occurring between two adjacent zones is the same for the ILP as for the heuristic.

Comparison of running times. We measured the running times of the heuristic and the ILP (see the last columns in Tables 6.2 and 6.3). Each running time includes the time for Steps 1–3. It excludes the running time for Steps 4–6, as these steps are not our contribution. The maximum running time of TwitterCircles of roughly one hour is due to the time limit we set for the ILP solver.

We observe a large difference between the ILP and the heuristic for both the average and maximum values. On the other hand, we can compute an optimal dual graph of Euler diagrams for the movie dataset within the time limit using our ILP. However, the heuristic is 138 times faster than the ILP on average. Setting a time limit of 10s for the ILP solver results in the same order of speedup for

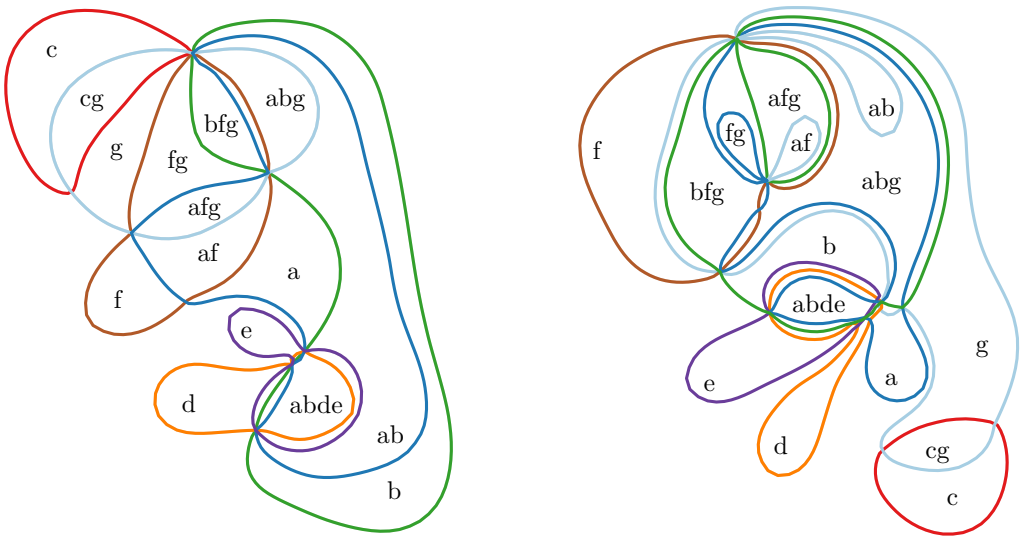


Figure 6.10: Resulting Euler diagrams for director Brett Ryan Bonowicz using the ILP (left) and the heuristic (right).

TwitterCircles. However, such a constrained time limit for the ILP solver leads to worse results than the heuristic.

Comparison with an existing method. Finally, we compared our exact method with the method by Rodgers et al. (2008), which allows sets to be split into multiple regions, breaking the duplicate curve label wellformedness condition as well as drawing layouts with concurrency, see Section 2.2. It is the closest state-of-the-art to our method. First, using our exact method, we identified all set systems in MovieDB that do not admit an Euler diagram with contiguous regions and without concurrency, yielding 187 set systems. For each of them, we used the method of Rodgers et al. (2008) to generate an Euler diagram. In 18.2% of the cases, the resulting Euler diagram splits at least one set into two or more regions. In contrast, in every case, our exact method with $\alpha = 0.01$ and $\beta = 0.1$ yielded an Euler diagram with a single contiguous region for each set and without losing a set element. Moreover, on average over all 187 set systems, our ILP reduced the concurrency by 35.35%.

6.6 Conclusion

We have proposed a novel ILP approach for finding a planar support of a given hypergraph that represents a set system. We have integrated this approach into a complete workflow for generating Euler diagrams that represent each set as a single contiguous region. The ILP maximizes the preserved number of set elements or set memberships. Additionally, it minimizes the total number of concurrent curves and places a maximum number of faces contained in only few curves adjacent to the outer face. Moreover, we have developed a heuristic that tries to optimize the same objective by greedily adding vertices to an empty graph. Our experiments show that our ILP can be used to produce optimal general Euler diagrams that comply with our wellformedness criteria. Moreover, our approach can simplify a set system in order to reduce concurrency. Comparisons of the heuristic with the ILP show that the heuristic produces results with similar objective values while increasing the concurrency by a factor of two. A benefit of the heuristic is that it needs only a fraction of the processing time.

For future work, producing Euler diagrams without concurrency by either limiting the edges within the superdual graph or merging sets might be possible. When concurrency cannot be avoided, equally distributing concurrency across edges instead of accumulating the total concurrency in a single edge is favorable. To improve readability, we could consider additional wellformedness conditions, such as the avoidance of triple points. Regarding the heuristic, the computation of candidates can be improved to avoid the brute force approach. As new ver-

tices are added to the dual graph, the algorithm must update existing candidates. Additionally, excluding certain edges from candidate creation might be beneficial (Wageningen et al., 2023). Our ILP could be accelerated, e.g., by generating multiple Kuratowski subdivisions at once (Chimani et al., 2019), instead of one by one. It may also be possible to explore ways of displaying removed set elements and indicating the lost information using visual mechanisms, such as texture, shading or icons. Furthermore, we would like to encode the degree of simplification and completeness of the Euler diagram using visualizations for quantitative information, such as area proportional layouts (Stapleton et al., 2011) or applying a color scale (Schloss et al., 2018).

Chapter 7

Conclusion and Future Work

In this thesis, we addressed problems in the field of map generalization, set system visualization, and set system simplification with combinatorial approaches influenced by spatial unit allocation. A key aspect of spatial unit allocation is the generation of compact results. We integrated the compactness requirement into the aggregation of polygons for map generalization and the grid-based drawing of Euler-like diagrams. To find a set of solutions that contains an optimal aggregation for each parameter value, we presented an approach that produces hierarchically nested results. We used another key aspect of spatial unit allocation, connectivity, to develop a novel approach for deciding whether a set system can be drawn as an Euler diagram while forming connected regions. In the same approach, we introduced a simplification technique that removes the least important set elements to reduce the complexity of the visualization. In the following, we will summarize the contributions for each research field of this thesis and discuss future work.

7.1 Summary of the Contributions

Map generalization. We presented an efficient approach for aggregating a set of polygons into larger groups of polygons. We aggregated polygons optimizing a trade-off between the total area and the perimeter of the solutions resulting in compact polygon groups. The area-perimeter trade-off is controlled by a single parameter λ. By transforming the problem into a graph representation and solving it with graph cuts, we are able to solve the aggregation within subquadratic time complexity. Furthermore, we showed that we can compute a linear-sized solution set that contains an optimal solution for every λ in the defined range of values. The resulting solution set is hierarchically structured, which allows for continuous aggregation while zooming on a map. Due to the large number of solutions with many small deviations from each other, we proposed an approxima-

tion algorithm for the solution set. In our experiments, we show that computing solutions with a narrow range of λ values is sufficient to produce results for a given map scale.

Set system visualization. We proposed a novel visualization approach called MosaicSets. The approach visualizes set systems as Euler-like diagrams in a grid-based layout. The only requirement for the set system is that each set element occurs exactly once in a set of the base map of the diagram. An important property of the resulting diagram is the contiguity of each set. Our conducted expert interviews indicated a need for such visualizations which are created automatically as creating a single visualization manually can take many hours. We proved that the problem of assigning the set elements to grid cells is NP-hard and therefore proposed an ILP approach solving the visualization task. Due to the complexity of the problem, we proposed an iterative approach that maximizes the compactness of each set in each iteration. In the experiments, our proposed iterative strategy performed significantly faster in practice without sacrificing quality of the results.

Set system simplification. We presented an exact algorithm to determine if a given set system can be drawn as an Euler diagram, in which every set has to form a single, connected region. Due to the interchangeability of set systems and hypergraphs, our algorithm is also able to determine whether a given hypergraph has a planar support. To the best of our knowledge, we proposed the first algorithm that can solve this problem optimally. Due to the NP-hardness of the problem, we proposed an ILP approach for computing exact solutions. We have formulated three objectives that determine the resulting dual graph of the Euler diagram. We applied our algorithm to numerous set systems and showed that it outperforms previous methods in terms of concurrency. When a set system cannot be drawn as an Euler diagram, we proposed a simplification technique that removes the least important set elements. The simplification is based on a weighted sum of objectives and produces results of maximum weight. This leads to the removal of the least important set elements and reduces the complexity of the visualization.

7.2 Future Work

In the following, we will discuss the future work and possible extensions of our approaches. Furthermore, we want to mention research which builds on our work and can be used to improve the results of our approaches.

Map generalization. Aggregating large sets of polygons into multiple groups of polygons remains a challenging task using our approach. We used the hierarchical property of the solution set to compute the linear-sized solution set. Further research can exploit the fact that each polygon group does not interact with other groups for larger λ values. Thus, the problem can be decomposed into smaller subproblems. Following up on our work, Beines et al. (2025) introduced a new minimum-cut approach to such problems that takes the hierarchical structure into account. Although they did not improve the theoretical computational complexity, they showed that their approach can compute all solutions within a reasonable time for large data sets. However, their approach loses the ability to approximate the solution set.

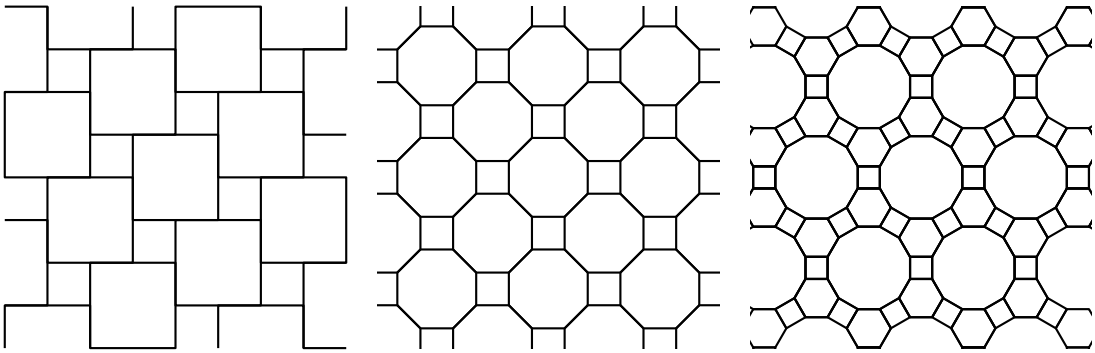
Another area of research is the efficient computation of bicriteria shapes for subdivisions which are based on elongated polygon edges. In our work, we showed that the complexity with respect to the number of input polygons is $O(n^6)$. For the case of building footprints, many narrow polygons with small sizes are constructed when using the extended edges. Further research is needed to reduce the number of polygons to a meaningful core set which aligns with the original building footprints.

A limitation of our work is that we only select the predefined polygons consisting of straight lines for the aggregation. Theoretically, the objective value of a solution with a fixed λ can be improved by allowing curved line segments. Such line segments depend on the parameter λ and need to be computed for each solution at the time of solving the solution. Although it is possible to improve the objective value, the use of curved line segments is currently not an established approach in cartography for map generalization (Wu and Wang, 2021).

As a future work, the bicriteria shapes can be extended to allow removal of polygons contained in the input polygons to be excluded from the output. Currently, every polygon contained in the input set is also contained in the output set. Removal of remaining small polygons from the output set is done in post-processing. Discarding portions of buildings is a common approach when simplifying building footprints (Haunert and Wolff, 2010b). It can be investigated whether bicriteria shapes are able to produce a similar simplification when using extended polygon boundaries. Thus, the bicriteria shapes could be extended from a generalization operator for aggregating polygons to an operator for selecting, simplifying and aggregating polygons.

Set system visualization. Although MosaicSets can produce visualizations of large set systems, a current limitation is that we cannot compute a visualization for every large set system. Also, we cannot determine in advance whether it is possible to compute it within a reasonable running time. Additionally, the

feasibility of the solution is dependent on the underlying grid graph. When using a grid graph with more adjacent cells, i.e. hexagonal over rectangular grid, the number of possible solutions increases. In the context, additional research on the choice of the grid graph is needed. An option is to use a grid graph which combine faces with more than six edges with faces which have less than six edges. We show three possible grid graphs in Figure 7.1. When using grid graphs combining different types of faces, a user study could determine whether the resulting diagrams lead to a higher cognitive load for the user.



(a) Pythagorean tiling (b) octagonal tiling (c) dodecagon-based tiling

Figure 7.1: Different types of tilings that can be used for the grid graph of MosaicSets. Each of the tilings is periodic, which allows a regular grid graph. Using faces with more than six edges increases the number of possible neighbors and helps to solve larger set systems.

Set system simplification. The generation of optimal Euler diagrams for arbitrary set systems opens up several possibilities for future work. One possible research direction is the use of our diagram construction in an interactive setup. Such an application can allow the user to select certain faces of the Euler diagram and expand them to show the details of the set elements. Currently, only the face without additional elements is displayed. However, adding additional elements to the face is part of the layout process and can be added when constructing the diagram.

Interactive Euler diagrams can be used to visualize the results of a literature search based on keywords. Current literature search tools build a knowledge graph of papers that cite or are cited by a query paper (Ammar et al., 2018; Weishuhn, 2025). In future research, we can use Euler diagrams to improve the visualization by using the keywords of a paper as sets. By using Euler diagrams to visualize the relationships between papers based on keywords, users can quickly grasp the similarity of papers and their relationships.

Another area of research is the use of our approach for constrained layouts of Euler diagrams, e.g. drawing sets as rectangles or circles. In current approaches, the layout of such Euler diagrams is done heuristically and often contains empty

faces. Until now, it is not known whether an Euler diagram with such a restricted layout exists and how to compute it (Priss and Dürrschnabel, 2024). Furthermore, we can investigate whether the approach can be extended to avoid or minimize triple points in the resulting Euler diagram. On the one hand, triple points can be reduced by using a layout generation algorithm that minimizes the number of triple points. However, some generated layouts will produce more triple points than others. Therefore, an additional objective function in the ILP can be used to minimize the number of triple points.

Adding additional constraints to the ILP regarding the planarity of the resulting dual graph can be used to reduce the number of callbacks in the ILP and potentially decrease the running time of the ILP-solver. A well-established property in cartography is that polygons in a map can be colored using four colors (Appel and Haken, 1989). In our case, the faces of the Euler diagram are analogous to the regions of a map. Therefore, the vertices of the dual graph can be colored with four colors. Although computing a 4-coloring of a graph is NP-complete (Dailey, 1980), it can be investigated whether the running time of the ILP can be improved by incorporating a 4-coloring formulation into the ILP. However, the 4-coloring of a graph is not a sufficient condition for planarity. Thus, a 4-coloring formulation cannot completely replace our introduced planarity constraint.

On the algorithm engineering side, the heuristic approach can be improved to be able to compute even larger set systems. Currently, the heuristic struggles to produce results for set systems consisting of more than 500 sets. On the one hand, the heuristic can be improved by updating existing candidates instead of generating all candidates in every iteration. This change can drastically reduce the time complexity of the heuristic.

Bibliography

Mikkel Abrahamsen, Panos Giannopoulos, Maarten Löffler, and Günter Rote. Geometric multicut: Shortest fences for separating groups of objects in the plane. *Discrete & Computational Geometry*, 64(3):575–607, October 2020. doi:10.1007/s00454-020-00232-w.

Jeroen C. J. H. Aerts and Gerard B. M. Heuvelink. Using simulated annealing for resource allocation. *International Journal of Geographical Information Science*, 16(6):571–587, 2002. doi:10.1080/13658810210138751.

Jeroen C. J. H. Aerts, Erwin Eisinger, Gerard B. M. Heuvelink, and Theodor J. Stewart. Using linear integer programming for multi-site land-use allocation. *Geographical Analysis*, 35(2):148–169, 2003. doi:10.1111/j.1538-4632.2003.tb01106.x.

Basak Alper, Nathalie Riche, Gonzalo Ramos, and Mary Czerwinski. Design study of linesets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011. doi:10.1109/TVCG.2011.186.

Bilal Alsallakh, Luana Micallef, Wolfgang Aigner, Helwig Hauser, Silvia Miksch, and Peter J. Rodgers. The state-of-the-art of set visualization. *Computer Graphics Forum*, 35(1):234–260, 2016. doi:10.1111/cgf.12722.

Waleed Ammar, Dirk Groeneveld, Chandra Bhagavatula, Iz Beltagy, Miles Crawford, Doug Downey, Jason Dunkelberger, Ahmed Elgohary, Sergey Feldman, Vu Ha, Rodney Kinney, Sebastian Kohlmeier, Kyle Lo, Tyler Murray, Hsu-Han Ooi, Matthew Peters, Joanna Power, Sam Skjonsberg, Lucy Lu Wang, Chris Wilhelm, Zheng Yuan, Madeleine van Zuylen, and Oren Etzioni. Construction of the literature graph in semantic scholar. In *Proc. 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 3, pages 84–91. Association for Computational Linguistics, June 2018. doi:10.18653/v1/N18-3011.

Karl-Heinrich Anders and Monika Sester. Parameter-free cluster detection in spatial databases and its application to typification. In *Proc. 19th ISPRS*

Congress, volume 33 of *International Archives of Photogrammetry and Remote Sensing*, pages 75–83, 2000.

Kenneth I Appel and Wolfgang Haken. *Every planar map is four colorable*, volume 98 of *Contemporary Mathematics*. American Mathematical Society, 1989. doi:10.1090/conm/098.

Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV). Ausleitung des ATKIS-Objektartenkataloges DLM250, 2022. URL `https://www.adv-online.de/icc/extdeu/nav/35b/binarywriterservlet?imgUid=a8130147-1420-4e71-0832-12914a39df1f&uBasVariant=11111111-1111-1111-1111-111111111111`. Accessed May 2025.

Daniel Barath and Jiří Matas. Graph-cut ransac. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, June 2018. doi:10.1109/CVPR.2018.00704.

Arne Beines, Michael Kaibel, Philip Mayer, Petra Mutzel, and Jonas Sauer. A simpler approach for monotone parametric minimum cut: Finding the breakpoints in order. In *Proc. Symposium on Algorithm Engineering and Experiments*, pages 29–41, 2025. doi:10.1137/1.9781611978339.3.

Michael A. Bekos, D.J.C. Dekker, F. Frank, Wouter Meulemans, Peter Rodgers, André Schulz, and S. Wessel. Computing schematic layouts for spatial hypergraphs on concentric circles and grids. *Computer Graphics Forum*, 2022. doi:10.1111/cgf.14497.

Bonnie Berkowitz and Lazaro Gamio. What you need to know about the measles outbreak. `https://www.washingtonpost.com/graphics/health/how-fast-does-measles-spread/`, 02 2015. Accessed May 2025.

Alain Billionnet. Mathematical optimization ideas for biodiversity conservation. *European Journal of Operational Research*, 231(3):514–534, 2013. doi:10.1016/j.ejor.2013.03.025.

Michael Bleyer and Margrit Gelautz. Graph-cut-based stereo matching using image segmentation with symmetrical treatment of occlusions. *Signal Processing: Image Communication*, 22(2):127–143, 2007. doi:10.1016/j.image.2006.11.012.

Fritz Bökler and Petra Mutzel. Output-sensitive algorithms for enumerating the extreme nondominated points of multiobjective combinatorial optimization problems. In *Proc. 23rd Annual European Symposium on Algorithms*, pages 288–299. Springer, 2015. doi:10.1007/978-3-662-48350-3_25.

Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. Retrieving α-shapes and schematic polygonal approximations for sets of points within queried temporal ranges. In *Proc. 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 249–258, 2019. doi:10.1145/3347146.3359087.

John M. Boyer and Wendy J. Myrvold. On the cutting edge: Simplified O(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8 (3):241–273, 2004. doi:10.7155/jgaa.00091.

Yuri Boykov and Olga Veksler. Graph cuts in vision and graphics: Theories and applications. In *Handbook of Mathematical Models in Computer Vision*, chapter 5, pages 79–96. Springer, 2006. doi:10.1007/0-387-28831-7_5.

Ulrik Brandes, Sabine Cornelsen, Barbara Pampel, and Arnaud Sallaberry. Blocks of hypergraphs applied to hypergraphs and outerplanarity. In *Proc. Combinatorial Algorithms*, volume 6460 of *LNCS*, pages 201–211. Springer, 2011. doi:10.1007/978-3-642-19222-7_21.

Gary L Brase. Pictorial representations in statistical reasoning. *Applied Cognitive Psychology*, 23(3):369–381, 2009. doi:10.1002/acp.1460.

Richard Brath. Multi-attribute glyphs on Venn and Euler diagrams to represent data and aid visual decoding. In *Proc. 3rd International Workshop on Euler Diagrams*, volume 854 of *CEUR*, pages 122–129, 2012.

Sandra M. Brown, Julie O. Culver, Kathryn E. Osann, Deborah J. MacDonald, Sharon Sand, Andrea A. Thornton, Marcia Grant, Deborah J. Bowen, Kelly A. Metcalfe, Harry B. Burke, Mark E. Robson, Susan Friedman, and Jeffrey N. Weitzel. Health literacy, numeracy, and interpretation of graphical breast cancer risk estimates. *Patient Education and Counseling*, 83(1):92–98, 2011. doi:10.1016/j.pec.2010.04.027.

Adrien Brunel, Jérémy Omer, Antoine Gicquel, and Sophie Lanco. Designing compact, connected and gap-free reserves with systematic reserve site selection models. *Applied Mathematical Modelling*, 134:307–323, 2024. doi:10.1016/j.apm.2024.06.001.

Kevin Buchin, Marc J. Kreveld, van, Henk Meijer, Bettina Speckmann, and Kevin A.B. Verbeek. On planar supports for hypergraphs. *Journal of Graph Algorithms and Applications*, 15(4):533–549, 2011. doi:10.7155/jgaa.00237.

Dirk Burghardt and Alessandro Cecconi. Mesh simplification for building typification. *International Journal of Geographical Information Science*, 21(3): 283–298, 2007. doi:10.1080/13658810600912323.

Dirk Burghardt, Stefan Schmid, and Jantien Stoter. Investigations on cartographic constraint formalisation. In *Proc. 11th ICA Workshop on Generalisation and Multiple Representation*, 2007.

Rafael G. Cano, Kevin Buchin, Thom Castermans, Astrid Pieterse, Willem Sonke, and Bettina Speckmann. Mosaic drawings and cartograms. *Computer Graphics Forum*, 34(3):361–370, 2015. doi:10.1111/cgf.12648.

Felipe Caro, Takeshi Shirabe, Monique Guignard, and Andrés Weintraub. School redistricting: embedding GIS tools with integer programming. *Journal of Operational Research Society*, 55(8):836–849, 2004. doi:10.1057/palgrave.jors.2601729.

Thom Castermans, Mereke van Garderen, Wouter Meulemans, Martin Nöllenburg, and Xiaoru Yuan. Short plane supports for spatial hypergraphs. *Journal of Graph Algorithms and Applications*, 23(3):463–498, 2019. doi:10.7155/jgaa.00499.

Xinjian Chen and Lingjiao Pan. A survey of graph cuts/graph search based medical image segmentation. *IEEE Reviews in Biomedical Engineering*, 11: 112–124, 2018. doi:10.1109/RBME.2018.2798701.

Markus Chimani, Thomas C. van Dijk, and Jan-Henrik Haunert. How to eat a graph: Computing selection sequences for the continuous generalization of road networks. In *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 243–252. Association for Computing Machinery, 2014. doi:10.1145/2666310.2666414.

Markus Chimani, Ivo Hedtke, and Tilo Wiedera. Exact algorithms for the maximum planar subgraph problem: New models and experiments. *ACM Journal of Experimental Algorithmics*, 24, April 2019. doi:10.1145/3320344.

Stirling Chow and Frank Ruskey. Towards a general solution to drawing area-proportional Euler diagrams. *Electronic Notes in Theoretical Computer Science*, 134:3–18, 2005. doi:10.1016/j.entcs.2005.02.017.

Marek Chrobak and Thomas H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995. doi:10.1016/0020-0190(95)00020-D.

Xiao Bao Clark, Jackson G. Finlay, Andrew J. Wilson, Keith L. J. Milburn, Minh Hoang Nguyen, Christof Lutteroth, and Burkhard C. Wünsche. An investigation into graph cut parameter optimisation for image-fusion applications. In *Proc. 27th Conference on Image and Vision Comput-*

ing New Zealand, pages 480–485. Association for Computing Machinery, 2012. doi:10.1145/2425836.2425927.

Jared L. Cohon. *Multiobjective Programming and Planning*. Academic Press, 1978. ISBN 9780080956497.

Christopher Collins, Gerald Penn, and Sheelagh Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009. doi:10.1109/TVCG.2009.122.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022. ISBN 9780262046305.

Azelle Courtial, Guillaume Touya, and Xiang Zhang and. Deriving map images of generalised mountain roads with generative adversarial networks. *International Journal of Geographical Information Science*, 37(3):499–528, 2023. doi:10.1080/13658816.2022.2123488.

David P. Dailey. Uniqueness of colorability and colorability of planar 4-regular graphs are NP-complete. *Discrete Mathematics*, 30(3):289–293, 1980. doi:10.1016/0012-365X(80)90236-8.

Jonathan Damen, Marc van Kreveld, and Bert Spaan. High quality building generalization by extending the morphological operators. In *Proc. 11th ICA Workshop on Generalisation and Multiple Representation*, pages 1–12, 2008.

Steven J. D'Amico, Shoou-Jiun Wang, Rajan Batta, and Christopher M. Rump. A simulated annealing approach to police district design. *Computers & Operations Research*, 29(6):667–684, 2002. doi:10.1016/S0305-0548(01)00056-9.

Constantinos Daskalakis, Ilias Diakonikolas, and Mihalis Yannakakis. How good is the chord algorithm? *SIAM Journal on Computing*, 45(3):811–858, 2016. doi:10.1137/13093875X.

Kasper Dinkla, Marc van Kreveld, Bettina Speckmann, and Michel A. Westenberg. Kelp diagrams: Point set membership visualization. *Computer Graphics Forum*, 31(3):875–884, 2012. doi:10.1111/j.1467-8659.2012.03080.x.

David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. Efficient generation of simple polygons for characterizing the shape of a set

of points in the plane. *Pattern Recognition*, 41(10):3224–3236, 2008. doi:10.1016/j.patcog.2008.03.023.

Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4): 551–559, 1983. doi:10.1109/TIT.1983.1056714.

Anthony William Fairbank Edwards. An eleventh-century venn diagram. *BSHM Bulletin: Journal of the British Society for the History of Mathematics*, 21(2): 119–121, 2006. doi:10.1080/17498430600804407.

Alon Efrat, Yifan Hu, Stephen G. Kobourov, and Sergey Pupyrev. MapSets: Visualizing embedded and clustered graphs. *Journal of Graph Algorithms and Applications*, 19(2):571–593, 2015. doi:10.7155/jgaa.00364.

Anton Ehrenzweig. *The Psycho-Analysis Of Artistic Vision And Hearing – An Introduction to a Theory of Unconscious Perception*. Routledge, 1953. doi:10.4324/9781315009230.

David Eppstein, Marc J. van Kreveld, Bettina Speckmann, and Frank Staals. Improved grid map layout by point set matching. *International Journal Computational Geometry and Applications*, 25(2):101–122, 2015. doi:10.1142/S0218195915500077.

Leonhard Euler. *Lettres à une princesse d'Allemagne: sur divers sujets de physique & de philosophie*. EPFL Press, 1768.

Yu Feng, Frank Thiemann, and Monika Sester. Learning cartographic building generalization with deep convolutional neural networks. *ISPRS International Journal of Geo-Information*, 8(6), 2019. doi:10.3390/ijgi8060258.

Jean Flower and John Howse. Generating Euler diagrams. In *Diagrammatic Representation and Inference*, pages 61–75. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-46037-3_6.

Lester Randolph Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956. doi:10.4153/CJM-1956-045-5.

Axel Forsch, Ruben Kemna, Elmar Langetepe, and Jan-Henrik Haunert. Polyline morphing for animated schematic maps. *Journal of Geovisualization and Spatial Analysis*, 8, 2024. doi:10.1007/s41651-024-00198-w.

Cheng Fu, Zhiyong Zhou, Yanan Xin, and Robert Weibel. Reasoning cartographic knowledge in deep learning-based map generalization with explainable AI. *International Journal of Geographical Information Science*, 38(10):2061–2082, 2024. doi:10.1080/13658816.2024.2369535.

Martin Galanda. *Automated polygon generalization in a multi agent system*. PhD thesis, University of Zurich, 2003.

Emden R. Gansner, Yifan Hu, and Stephen G. Kobourov. GMap: Drawing graphs as maps. In *Proc. Graph Drawing*, volume 5849 of *LNCS*, pages 405–407. Springer, 2009a. doi:10.1007/978-3-642-11805-0_38.

Emden R. Gansner, Yifan Hu, Stephen G. Kobourov, and Chris Volinsky. Putting recommendations on the map: visualizing clusters and relations. In *Proc. Recommender Systems*, pages 345–348. ACM, 2009b. doi:10.1145/1639714.1639784.

Peichao Gao, Haoyu Wang, Samuel A Cushman, Changxiu Cheng, Changqing Song, and Sijing Ye. Sustainable land-use optimization using NSGA-II: Theoretical and experimental comparisons of improved algorithms. *Landscape Ecology*, 36:1877–1892, 2021. doi:10.1007/s10980-020-01051-3.

Michael R. Garey, David S. Johnson, and Robert Endre Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5 (4):704–714, 1976. doi:10.1137/0205049.

Sven Gedicke, Johannes Oehrlein, and Jan-Henrik Haunert. Aggregating land-use polygons considering line features as separating map elements. *Cartography and Geographic Information Science*, 48(2):124–139, 2021. doi:10.1080/15230406.2020.1851613.

Jakob Geiger, Sabine Cornelsen, Jan-Henrik Haunert, Philipp Kindermann, Tamara Mchedlidze, Martin Nöllenburg, Yoshio Okamoto, and Alexander Wolff. ClusterSets: Optimizing planar clusters in categorical point data. *Computer Graphics Forum*, 40(3):471–481, 2021. doi:10.1111/cgf.14322.

Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.

Dorothy M. Greig, Bruce T. Porteous, and Allan H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society: Series B (Methodological)*, 51(2):271–279, 12 1989. doi:10.1111/j.2517-6161.1989.tb01764.x.

Günter Hake, Dietmar Grünreich, and Liqiu Meng. *Kartographie: Visualisierung raum-zeitlicher Informationen*. Walter de Gruyter, 2002. ISBN 978-3-11-016404-6.

Frank Harary. *Graph theory*. CRC Press, 1969. doi:10.1201/9780429493768.

Jan-Haunert Haunert and Alexander Wolff. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science*, 24(12):1871–1897, 2010a. doi:10.1080/13658810903401008.

Jan-Henrik Haunert and Alexander Wolff. Optimal simplification of building ground plans. In *Proc. 21st Congress of the International Society for Photogrammetry and Remote Sensing*, pages 373–378. International Society of Photogrammetry, Remote Sensing and Spatial Information Sciences, 2008.

Jan-Henrik Haunert and Alexander Wolff. Optimal and topologically safe simplification of building footprints. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 192–201. Association for Computing Machinery, 2010b. doi:10.1145/1869790.1869819.

Sidney W. Hess and Stuart A. Samuels. Experiences with a sales districting model: Criteria and implementation. *Management Science*, 18(4-part-ii):41–54, 1971. doi:10.1287/mnsc.18.4.P41.

Sidney W. Hess, James B. Weaver, H. J. Siegfeldt, Jillian N. Whelan, and Paul A. Zitlau. Nonpartisan political redistricting by computer. *Operations Research*, 13(6):998–1006, 1965. doi:10.1287/opre.13.6.998.

Marius Hogräfer, Magnus Heitzler, and Hans-Jörg Schulz. The state of the art in map-like visualization. *Computer Graphics Forum*, 39(3):647–674, 2020. doi:10.1111/cgf.14031.

Lina Huang, Tinghua Ai, Peter V. Oosterom, Xiongfeng Yan, and Min Yang. A matrix-based structure for vario-scale vector representation over a wide range of map scales: The case of river network data. *ISPRS International Journal of Geo-Information*, 6(7), 2017. doi:10.3390/ijgi6070218.

Ben Jacobsen, Markus Wallinger, Stephen G. Kobourov, and Martin Nöllenburg. MetroSets: Visualizing sets as metro maps. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1257–1267, 2021. doi:10.1109/TVCG.2020.3030475.

Mikael Jern, Jakib Rogstadius, and Tobias Åström. Treemaps and choropleth maps applied to regional hierarchical statistical data. In *Proc. 13th*

International Conference Information Visualisation, pages 403–410, 2009. doi:10.1109/IV.2009.97.

David S. Johnson and Henry O. Pollak. Hypergraph planarity and the complexity of drawing Venn diagrams. *Journal of Graph Theory*, 11(3):309–325, 1987. doi:10.1002/jgt.3190110306.

Christopher B. Jones, Geraint Ll. Bundy, and Mark J. Ware. Map generalization with a triangulated data structure. *Cartography and Geographic Information Systems*, 22(4):317–331, 1995. doi:10.1559/152304095782540221.

Michael Kaufmann, Marc van Kreveld, and Bettina Speckmann. Subdivision drawings of hypergraphs. In *Proc. Graph Drawing*, volume 5417 of *LNCS*, pages 396–407. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00219-9_39.

Rebecca Kehlbeck, Jochen Görtler, Yunhai Wang, and Oliver Deussen. SPEULER: semantics-preserving Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):433–442, 2022. doi:10.1109/TVCG.2021.3114834.

Mary Kelly, Aidan Slingsby, Jason Dykes, and Jo Wood. Historical internal migration in Ireland. In *Proc. GIS Research UK*, 2013. URL `https://openaccess.city.ac.uk/id/eprint/2052/`.

Bohyoung Kim, Bongshin Lee, and Jinwook Seo. Visualizing set concordance with permutation matrices and fan diagrams. *Interacting with Computers*, 19 (5-6):630–643, 2007. doi:10.1016/j.intcom.2007.05.004.

Kamyoung Kim, Denis J. Dean, Hyun Kim, and Yongwan Chun. Spatial optimization for regionalization problems with spatial interaction: a heuristic approach. *International Journal of Geographical Information Science*, 30(3): 451–473, 2016. doi:10.1080/13658816.2015.1031671.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, volume 4, pages 2713–2726, 2017. ISBN 9781713872719. doi:10.48550/arXiv.1609.02907.

Boris Klemz, Tamara Mchedlidze, and Martin Nöllenburg. Minimum tree supports for hypergraphs and low-concurrency Euler diagrams. In *Algorithm Theory*, volume 8503 of *LNCS*, pages 253–264. Springer, 2014. doi:10.1007/978-3-319-08404-6_23.

V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26 (2):147–159, 2004. doi:10.1109/TPAMI.2004.1262177.

Ephraim Korach and Michal Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98(1-3):385–414, 2003. doi:10.1007/S10107-003-0410-X.

Robert Kosara, TJ Jankun-Kelly, and Eleanor Chlan. IEEE InfoVis 2007 contest: InfoVis goes to the movies. `https://eagereyes.org/blog/2007/infovis-contest-2007-data`, 2007.

Casimir Kuratowski. Sur le probleme des courbes gauches en topologie. *Fundamenta mathematicae*, 15(1):271–283, 1930.

Harald Lachnit. The principle of contiguity. In *Principles of Learning and Memory*. Birkhäuser, 2006. doi:10.1007/978-3-0348-8030-5_1.

Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. Upset: visualization of intersecting sets. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1983–1992, 2014. doi:10.1109/TVCG.2014.2346248.

Chengming Li, Yong Yin, Xiaoli Liu, and Pengda Wu. An automated processing method for agglomeration areas. *ISPRS International Journal of Geo-Information*, 7(6):204, 2018. doi:10.3390/ijgi7060204.

Jingzhong Li and Tinghua Ai. A triangulated spatial model for detection of spatial characteristics of GIS data. In *Proc. IEEE International Conference on Progress in Informatics and Computing*, volume 1, pages 155–159, 2010. doi:10.1109/PIC.2010.5687417.

Xin Li and Lael Parrott. An improved genetic algorithm for spatial optimization of multi-objective and multi-site land use allocation. *Computers, Environment and Urban Systems*, 59:184–194, 2016. doi:10.1016/j.compenvurbsys.2016.07.002.

Erkki Mäkinen. How to draw a hypergraph. *International Journal of Computer Mathematics*, 34(3-4):177–185, 1990. doi:10.1080/00207169008803875.

Adrien Maudet, Guillaume Touya, Cécile Duchêne, and Sébastien Picault. Multi-agent multi-level cartographic generalisation in CartAGen. In *Proc. 12th International Conference on Advances in Practical Applications of Heterogeneous Multi-Agent Systems*, pages 355–358, 2014. doi:10.1007/978-3-319-07551-8_37.

Marc E. McDill, Stephanie A. Rebain, and Janis Braze. Harvest scheduling with area-based adjacency constraints. *Forest Science*, 48(4):631–642, November 2002. doi:10.1093/forestscience/48.4.631.

Robert B. McMaster and K. Stuart Shea. *Generalization in digital cartography*. Association of American Geographers, 1992. ISBN 9780892912094.

Graham McNeill and Scott A Hale. Generating tile maps. *Computer Graphics Forum*, 36:435–445, 2017. doi:10.1111/cgf.13200.

Wouter Meulemans, Nathalie Henry Riche, Bettina Speckmann, Basak Alper, and Tim Dwyer. KelpFusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013. doi:10.1109/TVCG.2013.76.

Wouter Meulemans, Jason Dykes, Aidan Slingsby, Cagatay Turkay, and Jo Wood. Small multiples with gaps. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):381–390, 2016. doi:10.1109/TVCG.2016.2598542.

Wouter Meulemans, Max Sondag, and Bettina Speckmann. A simple pipeline for coherent grid maps. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1236–1246, 2021. doi:10.1109/TVCG.2020.3028953.

Luana Micallef and Peter Rodgers. eulerForce: Force-directed layout for Euler diagrams. *Journal of Visual Languages and Computing*, 25(6):924–934, 2014. doi:10.1016/j.jvlc.2014.09.002.

Luana Micallef, Pierre Dragicevic, and Jean-Daniel Fekete. Assessing the effect of visualizations on Bayesian reasoning through crowdsourcing. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2536–2545, 2012. doi:10.1109/TVCG.2012.199.

Dimitrios Michail, Joris Kinable, Barak Naveh, and John V. Sichi. JGraphT—A Java library for graph data structures and algorithms. *ACM Transactions on Mathematical Software*, 46(2), 2020. doi:10.1145/3381449.

Adriano Moreira and Maribel Y. Santos. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *Proc. 2nd International Conference on Computer Graphics Theory and Applications*, pages 61–68, 2007. doi:10.5220/0002080800610068.

Darek J. Nalle, Jeffrey L. Arthur, and John Sessions. *Designing Compact and Contiguous Reserve Networks under a Budget Constraint*, volume 7 of *Managing Forest Ecosystems*, pages 243–247. Springer Netherlands, 2003. doi:10.1007/978-94-017-0307-9_23.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley interscience series in discrete mathematics and optimization. Wiley, 1988. doi:10.1002/9781118627372.

Lucas Nunes, Xieyuanli Chen, Rodrigo Marcuzzi, Aljosa Osep, Laura Leal-Taixé, Cyrill Stachniss, and Jens Behley. Unsupervised class-agnostic instance segmentation of 3D lidar data for autonomous vehicles. *IEEE Robotics and Automation Letters*, 7(4):8713–8720, 2022. doi:10.1109/LRA.2022.3187872.

Martin Nöllenburg, Damian Merrick, Alexander Wolff, and Marc Benkert. Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems*, 32(4):248–260, 2008. doi:10.1016/j.compenvurbsys.2008.06.004.

Johannes Oehrlein and Jan-Henrik Haunert. A cutting-plane method for contiguity-constrained spatial aggregation. *Journal of Spatial Information Science*, 15(1):89–120, 2017. doi:10.5311/JOSIS.2017.15.379.

Peter Oliver, Eugene Zhang, and Yue Zhang. Scalable hypergraph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 30(1):595–605, 2024. doi:10.1109/TVCG.2023.3326599.

OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2020.

James B. Orlin. Max flows in $O(nm)$ time, or better. In *Proc. 45th Annual ACM Symposium on Theory of Computing*, pages 765–774. Association for Computing Machinery, 2013. doi:10.1145/2488608.2488705.

Patrick Paetzold, Rebecca Kehlbeck, Hendrik Strobelt, Yumeng Xue, Sabine Storandt, and Oliver Deussen. RectEuler: Visualizing intersecting sets using rectangles. *Computer Graphics Forum*, 42(3):87–98, 2023. doi:10.1111/cgf.14814.

Stephen Palmer and Irvin Rock. Rethinking perceptual organization: The role of uniform connectedness. *Psychonomic Bulletin & Review*, 1:29–55, 1994. doi:10.3758/BF03200760.

Bo Peng and Olga Veksler. Parameter selection for graph cut based image segmentation. In *Proc. British Machine Vision Conference*, pages 16.1–16.10, 2008. doi:10.5244/C.22.16.

Bo Peng, Lei Zhang, and David Zhang. A survey of graph theoretical approaches to image segmentation. *Pattern Recognition*, 46(3):1020–1038, 2013. doi:10.1016/j.patcog.2012.09.015.

Dongliang Peng and Guillaume Touya. Continuously generalizing buildings to built-up areas by aggregating and growing. In *Proc. 3rd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, UrbanGIS'17. Association for Computing Machinery, 2017. doi:10.1145/3152178.3152188.

Dongliang Peng, Alexander Wolff, and Jan-Henrik Haunert. Finding optimal sequences for area aggregation—A* vs. integer linear programming. *ACM Transactions on Spatial Algorithms and Systems*, 7(1), October 2020. doi:10.1145/3409290.

Ekaterina S. Podolskaya, Karl-Heinrich Anders, Jan-Henrik Haunert, and Monika Sester. Quality assessment for polygon generalization. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVI-2/C43, 2007.

Daniel D. Polsby and Robert D. Popper. The third criterion: Compactness as a procedural safeguard against partisan gerrymandering. *Yale Law & Policy Review*, 9:301–353, 1991.

Melanie Price, Rachel Cameron, and Phyllis Butow. Communicating risk information: the influence of graphical display format on quantitative information perception—accuracy, comprehension and preferences. *Patient education and counseling*, 69(1-3):121–128, 2007. doi:10.1016/j.pec.2007.08.006.

Uta Priss and Dominik Dürrschnabel. Rectangular Euler diagrams and order theory. In *Diagrammatic Representation and Inference*, volume 14981, pages 165–181. Springer, 2024. doi:10.1007/978-3-031-71291-3_14.

Anthony Przybylski, Xavier Gandibleux, and Matthias Ehrgott. A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme. *Journal on Computing*, 22(3):371–386, 2010. doi:10.1287/ijoc.1090.0342.

Rob Radburn. Go with the flow: Commuting & migration flows within London. `https://public.tableau.com/app/profile/robradburn/viz/ODMpasLondonAftertheFlood/GowiththeFlow`, March 2016. Accessed August 2024.

Md. Mostafizur Rahman and György Szabó. Multi-objective urban land use optimization using spatial data: A systematic review. *Sustainable Cities and Society*, 74:103214, 2021. doi:10.1016/j.scs.2021.103214.

Rajiv Raman and Saurabh Ray. Planar Support for Non-piercing Regions and Applications. In *Proc. 26th Annual European Symposium on Algorithms*, volume 112 of *LIPIcs*, pages 69:1–69:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.69.

Federica Ricca and Bruno Simeone. Local search algorithms for political districting. *European Journal of Operational Research*, 189(3):1409–1426, 2008. doi:10.1016/j.ejor.2006.08.065.

Nathalie Henry Riche and Tim Dwyer. Untangling Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010. doi:10.1109/TVCG.2010.210.

Phillipe Rigaux, Michel Scholl, and Agnés Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2002. doi:10.1016/B978-1-55860-588-6.X5000-3.

Roger Z. Ríos-Mercado, editor. *Optimal Districting and Territory Design*. Springer, 2020. doi:10.1007/978-3-030-34312-5.

Peter Rodgers. A survey of Euler diagrams. *Journal of Visual Languages & Computing*, 25(3):134–155, 2014. doi:10.1016/j.jvlc.2013.08.006.

Peter Rodgers, Leishi Zhang, and Andrew Fish. General Euler diagram generation. In *Diagrammatic Representation and Inference*, volume 5223 of *LNCS*, pages 13–27. Springer, 2008. doi:10.1007/978-3-540-87730-1_6.

Peter Rodgers, Leishi Zhang, and Helen Purchase. Wellformedness properties in Euler diagrams: Which should be used? *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1089–1100, 2011. doi:10.1109/TVCG.2011.143.

Leon Rosenberger, Yilang Shen, and Jan-Henrik Haunert. Simultaneous selection and displacement of buildings and roads for map generalization via mixed-integer quadratic programming. *International Journal of Geographical Information Science*, pages 1–30, 2025. doi:10.1080/13658816.2025.2461602.

Robert E. Roth, Michael Stryker, and Cynthia A. Brewer. A typology of multiscale mapping operators. In *Proc. 5th International Conference on Geographic Information Science*, number 1996 in Leibniz International Proceedings in Informatics (LIPIcs), 2008.

Peter Rottmann, Anne Driemel, Herman Haverkort, Heiko Röglin, and Jan-Henrik Haunert. Bicriteria Aggregation of Polygons via Graph Cuts. In *Proc. 11th International Conference on Geographic Information Science - Part II*, volume 208 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:16, 2021. doi:10.4230/LIPIcs.GIScience.2021.II.6.

Peter Rottmann, Markus Wallinger, Annika Bonerath, Sven Gedicke, Martin Nöllenburg, and Jan-Henrik Haunert. MosaicSets: Embedding set systems into grid graphs. *IEEE Transactions on Visualization and Computer Graphics*, 29(1):875–885, 2023. doi:10.1109/TVCG.2022.3209485.

Peter Rottmann, Peter Rodgers, Xinyuan Yan, Daniel Archambault, Bei Wang, and Jan-Henrik Haunert. Generating Euler diagrams through combinatorial optimization. *Computer Graphics Forum*, 43(3), 2024. doi:10.1111/cgf.15089.

Peter Rottmann, Anne Driemel, Herman Haverkort, Heiko Röglin, and Jan-Henrik Haunert. Bicriteria Shapes: Hierarchical grouping and aggregation of polygons with an efficient graph-cut approach. *ACM Transactions on Spatial Algorithms and Systems*, 11(1):1–23, February 2025. doi:10.1145/3705001.

Ramik Sadana, Timothy Major, Alistair Dove, and John Stasko. Onset: A visualization technique for large-scale binary set data. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):1993–2002, 2014. doi:10.1109/TVCG.2014.2346249.

Azimjon Sayidov, Robert Weibel, and Stefan Leyk. Recognition of group patterns in geological maps by building similarity networks. *Geocarto International*, 37 (2):607–626, 2022. doi:10.1080/10106049.2020.1730449.

Jochen Schiewe. Distortion effects in equal area unit maps. *KN – Journal of Cartography and Geographic Information*, 71(2):71–82, 2021. doi:10.1007/s42489-021-00072-5.

Karen B. Schloss, Connor C. Gramazio, Allison T. Silverman, Madeline L. Parker, and Audrey S. Wang. Mapping color to meaning in colormap data visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1): 810–819, 2018. doi:10.1109/TVCG.2018.2865147.

Nadine Schwartges, Dennis Allerkamp, Jan-Henrik Haunert, and Alexander Wolff. Optimizing active ranges for point selection in dynamic maps. In *Proc. the 16th ICA Generalisation Workshop*, 2013.

Joseph E. Schwartzberg. Reapportionment, gerrymanders, and the notion of compactness. *Minnesota Law Review*, 50:443–452, 1965.

David Sedlacek and Jiri Zara. Graph cut based point-cloud segmentation for polygonal reconstruction. In *Proc. 5th International Symposium on Advances in Visual Computing*, volume 5876 of *LNCS*, pages 218–227. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-10520-3_20.

Monika Sester and Claus Brenner. Continuous generalization for visualization on small mobile devices. In *Developments in Spatial Data Handling*, pages 355–368. Springer Berlin Heidelberg, 2005. doi:10.1007/3-540-26772-7_27.

Monika Sester, Karl-Heinrich Anders, and Volker Walter. Linking objects of different spatial data sets by integration and aggregation. *GeoInformatica*, 2 (4):335–358, 1998. doi:10.1023/A:1009705404707.

Tim Shaw. Good data visualization practice: Tile grid maps. `https://forumone.com/ideas/good-data-visualization-practice-tile-grid-maps-0`, April 2016. Accessed March 2022.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000. doi:10.1109/34.868688.

Takeshi Shirabe. A model of contiguity for spatial unit allocation. *Geographical Analysis*, 37(1):2–16, 2005. doi:10.1111/j.1538-4632.2005.00605.x.

Takeshi Shirabe. Districting modeling with exact contiguity constraints. *Environment and Planning B: Planning and Design*, 36(6):1053–1066, 2009. doi:10.1068/b34104.

Paolo Simonetto and David Auber. Visualise undrawable Euler diagrams. In *Proc. 12th International Conference Information Visualisation*, pages 594–599, 2008. doi:10.1109/IV.2008.78.

Paolo Simonetto, David Auber, and Daniel Archambault. Fully automatic visualisation of overlapping sets. *Computer Graphics Forum*, 28(3):967–974, 2009. doi:10.1111/j.1467-8659.2009.01452.x.

Paolo Simonetto, Daniel Archambault, David Auber, and Romain Bourqui. ImPrEd: An improved force-directed algorithm that prevents nodes from crossing edges. *Computer Graphics Forum*, 30(3):1071–1080, 2011. doi:10.1111/j.1467-8659.2011.01956.x.

Paolo Simonetto, Daniel Archambault, and Carlos Scheidegger. A simple approach for boundary improvement of Euler diagrams. *IEEE Trans-*

actions on Visualization and Computer Graphics, 22(1):678–687, 2016. doi:10.1109/TVCG.2015.2467992.

Aidan Slingsby and Emiel van Loon. Temporal tile-maps for characterising the temporal occupancy of places: A seabird case study. In *Proc. GIS Research UK*, 2017. URL `https://openaccess.city.ac.uk/id/eprint/17398/1/`.

Gem Stapleton, Peter Rodgers, John Howse, and John Taylor. Properties of Euler diagrams. *Electronic Communications of the EASST*, 7, September 2007. doi:10.14279/tuj.eceasst.7.92.

Gem Stapleton, Peter Rodgers, and John Howse. A general method for drawing area-proportional Euler diagrams. *Journal of Visual Languages & Computing*, 22(6):426–442, 2011. doi:10.1016/j.jvlc.2011.07.001.

Stefan Steiniger, Dirk Burghardt, and Robert Weibel. Recognition of island structures for map generalization. In *Proc. 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, pages 67–74, 2006. doi:10.1145/1183471.1183484.

Radan Šuba. Design and development of a system for vario-scale maps. *Architecture and the Built Environment*, 7(18):1–162, 2017. doi:10.7480/abe.2017.18.

Radan Šuba, Martijn Meijers, Lina Huang, and Peter van Oosterom. *An Area Merge Operation for Smooth Zooming*, pages 275–293. LNCS. Springer, 2014. doi:10.1007/978-3-319-03611-3_16.

Fernando Tavares-Pereira, José Rui Figueira, Vincent Mousseau, and Bernard Roy. Multiple criteria districting problems. *Annals of Operations Research*, 154(1):69–92, 2007. doi:10.1007/s10479-007-0181-5.

Soulivanh Thao, Mats Garvik, Gregoire Mariethoz, and Mathieu Vrac. Combining global climate models using graph cuts. *Climate Dynamics*, 59(7):2345–2361, 2022. doi:10.1007/s00382-022-06213-4.

Sabine Timpf and Andrew U. Frank. A multi-scale data structure for cartographic objects. In *Proc. 17th International Cartographic Conference*, pages 1389 – 1396, 1995.

Daoqin Tong and Alan T. Murray. Spatial optimization in geography. *Annals of the Association of American Geographers*, 102(6):1290–1309, 2012. doi:10.1080/00045608.2012.685044.

Guillaume Touya, Xiang Zhang, and Imran Lokhat. Is deep learning the new agent for map generalization? *International Journal of Cartography*, 5(2–3): 142–157, 2019. doi:10.1080/23729333.2019.1613071.

Edward R. Tufte. *The visual display of quantitative information.* Graphics Press, 1992. doi:10.1119/1.14057.

Hamidreza Validi, Austin Buchanan, and Eugene Lykhovyd. Imposing contiguity constraints in political districting models. *Operations Research*, 70(2):867–892, 2021. doi:10.1287/opre.2021.2141.

René van Bevern, Iyad A. Kanj, Christian Komusiewicz, Rolf Niedermeier, and Manuel Sorge. Twins in subdivision drawings of hypergraphs. In *Proc. 24th International Symposium Graph Drawing and Network Visualization*, volume 9801 of *LNCS*, pages 67–80. Springer, 2016. doi:10.1007/978-3-319-50106-2_6.

René van Bevern, Iyad Kanj, Christian Komusiewicz, Rolf Niedermeier, and Manuel Sorge. The role of twins in computing planar supports of hypergraphs. *Journal of Graph Algorithms and Applications*, 28(1):51–79, May 2024. doi:10.7155/jgaa.v28i1.2927.

Marc Van Kreveld. Smooth generalization for continuous zooming. In *Proc. 20th International Geographic Conference*, pages 2180–2185, 2001.

Peter van Oosterom, Martijn Meijers, Jantien Stoter, and Radan Šuba. *Data Structures for Continuous Generalisation: tGAP and SSC*, pages 83–117. LNCS. Springer, 2014. doi:10.1007/978-3-319-00203-3_4.

John Venn. I. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 10(59):1–18, 1880. doi:10.1080/14786448008626877.

Anne Verroust and Marie-Luce Viaud. Ensuring the drawability of extended Euler diagrams for up to 8 sets. In *Diagrammatic Representation and Inference*, volume 2980 of *LNCS*, pages 128–141. Springer, 2004. doi:10.1007/978-3-540-25931-2_13.

Sara Vicente, Vladimir Kolmogorov, and Carsten Rother. Graph cut based image segmentation with connectivity priors. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. doi:10.1109/CVPR.2008.4587440.

Simon van Wageningen, Tamara Mchedlidze, and Alexandru Telea. Identifying Cluttering Edges in Near-Planar Graphs. In *EuroVis 2023 - Short Papers*. The Eurographics Association, 2023. doi:10.2312/evs.20231048.

Michael Weishuhn. Inciteful: Citation network exploration. https://inciteful.xyz, 2025. Accessed April 2025.

Krist Wongsuphasawat. A semi-automatic way to create your own grid map. https://medium.com/free-code-camp/creating-grid-map-for-thailand-397b53a4ecf, 2016. Accessed March 2022.

Jo Wood and Jason Dykes. Spatially ordered treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1348–1355, 2008. doi:10.1109/TVCG.2008.165.

Jo Wood, Jason Dykes, and Aidan Slingsby. Visualisation of origins, destinations and flows with OD maps. *The Cartographic Journal*, 47(2):117–129, 2010. doi:10.1179/000870410X12658023467367.

Jo Wood, Aidan Slingsby, and Jason Dykes. Visualizing the dynamics of London's bicycle-hire scheme. *Cartographica*, 46(4):239–251, 2011. doi:10.3138/carto.46.4.239.

Fang Wu and Jiayao Wang. *Cartographic Generalization*, pages 151–211. Springer Singapore, 2021. doi:10.1007/978-981-16-0614-4_5.

Zhenyu Wu and Richard Leahy. An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993. doi:10.1109/34.244673.

Ningchuan Xiao, Myung Jin Kim, and Yue Lin. A multistart and recombination algorithm for finding many unique solutions to spatial aggregation problems. *GeoInformatica*, 29:1–39, 2024a. doi:10.1007/s10707-024-00520-0.

Tianyuan Xiao, Tinghua Ai, Huafei Yu, Min Yang, and Pengcheng Liu and. A point selection method in map generalization using graph convolutional network model. *Cartography and Geographic Information Science*, 51(1):20–40, 2024b. doi:10.1080/15230406.2023.2187886.

Xinyuan Yan, Peter Rodgers, Peter Rottmann, Daniel Archambault, Jan-Henrik Haunert, and Bei Wang. EulerMerge: Simplifying Euler diagrams through set merges. In *Diagrammatic Representation and Inference*, volume 14981 of *LNCS*, pages 190–206. Springer, 2024. doi:10.1007/978-3-031-71291-3_16.

Mei Yang, Tong Chen, Yong-Xin Liu, and Luqi Huang. Visualizing set relationships: Evenn's comprehensive approach to venn diagrams. *iMeta*, 3(3):e184, 2024. doi:https://doi.org/10.1002/imt2.184.

Romano Zachary. Data visualization strategies using tile grid maps. https://www.gislounge.com/

`data-visualization-strategies-using-tile-grid-maps/`, November 2015. Accessed March 2022.

Lukas Zebedin, Joachim Bauer, Konrad Karner, and Horst Bischof. Fusion of feature- and area-based information for urban buildings modeling from aerial imagery. In *Computer Vision – ECCV 2008*, volume 5305 of *LNCS*, pages 873–886. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-88693-8_64.

Jack Zhang, Hamidreza Validi, Austin Buchanan, and Illya V Hicks. Linear-size formulations for connected planar graph partitioning and political districting. *Optimization Letters*, 18(1):19–31, 2024. doi:10.1007/s11590-023-02070-0.

Rong Zhao, Tinghua Ai, and Chen Wen. A method for generating variable-scale maps for small displays. *ISPRS International Journal of Geo-Information*, 9 (4), 2020. doi:10.3390/ijgi9040250.

Youjia Zhou, Archit Rathore, Emilie Purvine, and Bei Wang. Topological simplifications of hypergraphs. *IEEE Transactions on Visualization and Computer Graphics*, 29(7):3209–3225, 2023a. doi:10.1109/TVCG.2022.3153895.

Zhiyong Zhou, Cheng Fu, and Robert Weibel. Move and remove: Multi-task learning for building simplification in vector maps with a graph convolutional neural network. *ISPRS Journal of Photogrammetry and Remote Sensing*, 202: 205–218, 2023b. doi:10.1016/j.isprsjprs.2023.06.004.

List of Figures

1.1 An example of map generalization. 1
1.2 Different visualization techniques for set systems. 2
1.3 An Euler diagram visualizing treaties within Europe. 3
1.4 Aggregation of the polygons of the buildings near the Nussallee in Bonn. 6
1.5 An example of a MosaicSets visualization. 7

2.1 An α-shape that generates a narrow bridge between two point sets. 11
2.2 Examples of wellformedness conditions following Rodgers et al. (2011). 15

3.1 Examples of different graphs. 20
3.2 The non-planar, complete graphs K_5 and $K_{3,3}$. 21
3.3 The plane graph with its dual and adjacency graph. 23
3.4 A planar support of the hypergraph. 25
3.5 An Euler diagram whose dual graph is the planar support in Figure 3.4. 25
3.6 The planar support of the hypergraph in Figure 3.4 superimposed with an Euler diagram using the planar support as its dual graph. 25
3.7 Spatial unit allocation. 28
3.8 Graph cut model. 33

4.1 Input polygons aggregated to larger ones. 37
4.2 Algorithmic solution of Problem 1 via a graph cut. 39
4.3 Different types of solutions and Pareto-frontier of a bicriteria optimization problem. 42
4.4 Schematic visualization of two λ-optimal solutions. 42
4.5 Polygon with triangulation of the exterior space. 43
4.6 Set of solutions for the instance in Figure 4.5. 43
4.7 Geometrical representation of solutions. 44
4.8 Solutions obtained with Algorithm 1 for the same instance without (a) and with approximation (b). 47

4.9 Result set of two evaluation data sets with $\varepsilon = 0.1$. 48
4.10 Evaluation metrics for the settlement area of Euskirchen. 49
4.11 A large instance aggregated by our approach. 50
4.12 IoU-scores of multiple instances in relation to our λ-value. 51
4.13 Histogram of the λ values of the respective best solutions with respect to the IoU value. 52
4.14 Extending the boundary edges for given polygons to create edge-aligned polygons. 53
4.15 Ahrem aggregated with polygons created by extruding polygon edges. 54
4.16 Ahrem generated with our proposed approach and with an α-shape. 54
4.17 All α-shape solutions of instances shown in Figure 4.16 and Figure 4.11. 55

5.1 Visualizing the research groups of the Agricultural Faculty of the University of Bonn with *MosaicSets*. 59
5.2 Required input files and a solution using a basic rendering style. . 62
5.3 An instance and a solution of the basic ILP. 67
5.4 Different combinations of proposed rendering styles. 72
5.5 Manually generated visualization showing the research groups of the Agricultural Faculty of the University of Bonn. 74
5.6 MosaicSets computed with eccentricity-based compactness (MSE) for Bonn with different numbers of projects. 77
5.7 Comparison of the different compactness approaches for Vienna using three overlay sets 78

6.1 Results after different steps of our hypergraph visualization workflow. 85
6.2 Construction of the superdual graph. 86
6.3 The directed graph $\tilde{G}_X$ for a hyperedge X. 88
6.4 A feasible assignment of the variables modeling the connectivity of the selected subgraph for X. 88
6.5 Graph G' after adding three vertices and three edges with our heuristic. 92
6.6 Euler diagram workflow of set system corresponding to director Keith Hooker. 94
6.7 Simplifying the set system of director Joel Schoenbach from left to right. 94
6.8 Comparison of results for director Art Camacho for $\beta = 0.1$ and $\beta = 0$. 95
6.9 Comparison of concurrency of our heuristic and ILP approach. . . 97

6.10 Resulting Euler diagrams for director Brett Ryan Bonowicz using the ILP (left) and the heuristic (right). 97

7.1 Different types of tilings that can be used for the grid graph of MosaicSets. 104

A.1 Parliament with eight overlay sets visualized using MSE and relaxing the contiguity constraint as described in Section 4.3. . . . 134
A.2 Vienna (a) and Parliament (b) with three overlay sets visualized with the Kelp style. 134
A.3 MosaicSets visualization for Bonn with three overlay sets and a hexagonal grid. 135
A.4 MosaicSets visualization for Bonn with three overlay sets and a square grid. 135
A.5 MosaicSets visualization for Vienna with three overlay sets and a square grid. 136
A.6 Vienna with three overlay sets and square grid visualized with MosaicSets. 136
A.7 Parliament with three overlay sets and hexagonal grid visualized with MosaicSets. 137
A.8 Parliament with three overlay sets and square grid visualized with MosaicSets. 137

B.1 Euler diagram of the set system corresponding to the director Art Camacho from MovieDB. 149
B.2 Influence of parameter α on the set system corresponding to the director Leigh Slawner from MovieDB. 152
B.3 Influence of parameter α on the set system corresponding to the director Joel Schoenbach from MovieDB. 152
B.4 Influence of parameter α on the set system corresponding to the director Jim Wynorski from MovieDB. 153
B.5 Influence of parameter α on the set system corresponding to the director Art Camacho from MovieDB. 153
B.6 Influence of parameter β on the set system corresponding to the director Leigh Slawner from MovieDB. 153

List of Tables

2.1 A subset of generalization operators described by McMaster and Shea (1992). 10

6.1 Statistics of MovieDB and TwitterCircles. 93
6.2 Results of the ILP and the heuristic on MovieDB. 96
6.3 Results of the ILP and the heuristic on TwitterCircles. 96

A.1 Task taxonomy by Alsallakh et al. (2016) and assessment of MosaicSets, Euler diagrams and frequency grids (Micallef et al., 2012). 138

Appendix A

Supplemental Mosaic Sets

In this section, we provide supplemental material that is not included in the main manuscript due to the readability of the document. We show all visualizations of set systems for which our metrics are evaluated in the thesis.

A.1 Additional Figures

Figures for Section 5.4.3. Figure A.1 shows the proposed relaxation of the contiguity constraint which is explained in Section 5.4.3. In detail, we do not enforce the contiguity of the overlay sets anymore. We use Parliament for the comparison where we include eight overlay sets. When trying to solve this with the eccentricity-based approach MSE, we terminated the experiment after ten minutes. Until then MSE did not find a valid solution. Relaxing the contiguity constraint led to a solution within few seconds. The caption gives detailed running times.

Figures for Section 5.5. In Figure A.2, we illustrate examples for MosaicSets using the Kelp rendering style. We computed MosaicSets with MSE. The corresponding boundary style visualizations are given in this document in Figure A.5a and Figure A.8a, respectively.

Figures for Section 5.6.5 and Section 5.6.6. In Figure A.3 to Figure A.8, we compare visualizations of MosaicSets computed with MSE and MSEA. We recall that MSE uses an eccentricity-based compactness measure in all iterations. MSEA is a variant of MSE that restricts the available grid cells to those used in the first iteration and therefore confines the available grid area. In the captions of the figures, we also provide measures on compactness (mean Polsby-Popper score) and the running times. We show such a comparison for each data set with a hexagonal grid and afterwards with a square grid.

Bonn is shown in Figure A.3 and Figure A.4. The data set consists of 51 research groups, six departments and three research projects. We use the departments as sets of the base map and visualize the research projects as overlays.

Vienna is shown in Figure A.5 and Figure A.6. The data set consists of 71 research groups belonging to four different institutes and three research projects. Again the institutes are used as sets for the base map and the research projects are visualized as overlays.

Parliament is shown in Figures A.7 and A.8. The base map represents all 178 members of the Austrian parliament colored by their affiliation to one of the five political parties. The coloring of the base map uses lighter shades of the different parties' official colors. The overlay shows interest groups. Such a map could aid in transparency by understanding if a vote was cast to serve the interest of groups, networks, or individuals.

A.2 Task Taxonomy

Alsallakh et al. (2016) introduced a taxonomy of tasks commonly associated with set visualization techniques. Tasks are classified into three broad categories:

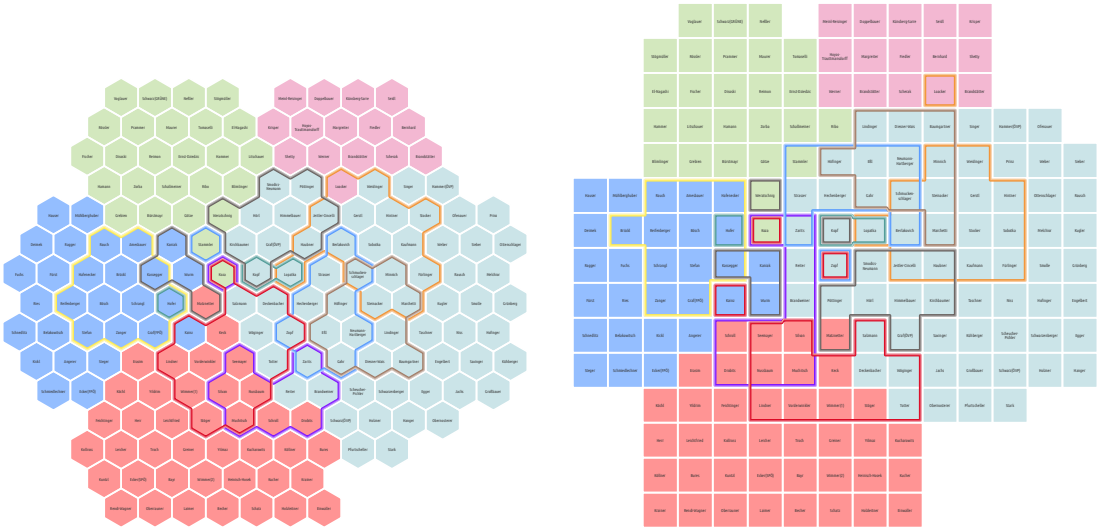
1. Tasks in Group **A** are element-based tasks that are concerned with elements and their respective relationship to the sets. For example: In which research projects is the research group 'Data Science in Agricultural Economics' involved in?
2. Tasks in Group **B** are related to sets and the relationship between different sets without taking individual elements into account. For example: Which research projects do overlap with project "PhenoRob"?
3. Tasks in Group **C** are related to element attributes and consider attributes of set elements and their relationship of distribution with regard to set membership. For example: Do research groups in the "PhenoRob" project publish more papers than research groups in the "DETECT" project?

MosaicSets only supports tasks in Groups A and B as we do not represent element attributes. Table A.1 shows all tasks of the two categories A and B. We list the assessment from our experts whether MosaicSets supports a task fully, partially or not at all. In order to compare MosaicSets to similar set visualization techniques, we also include Euler diagrams and frequency grids in Table A.1.

A.3 Expert Interviews

We also append the manuscript used for the expert interviews. We performed one preliminary interview in March 2022 with two experts. We only consider their opinions on the rendering style from this interview. Later, in June 2022,

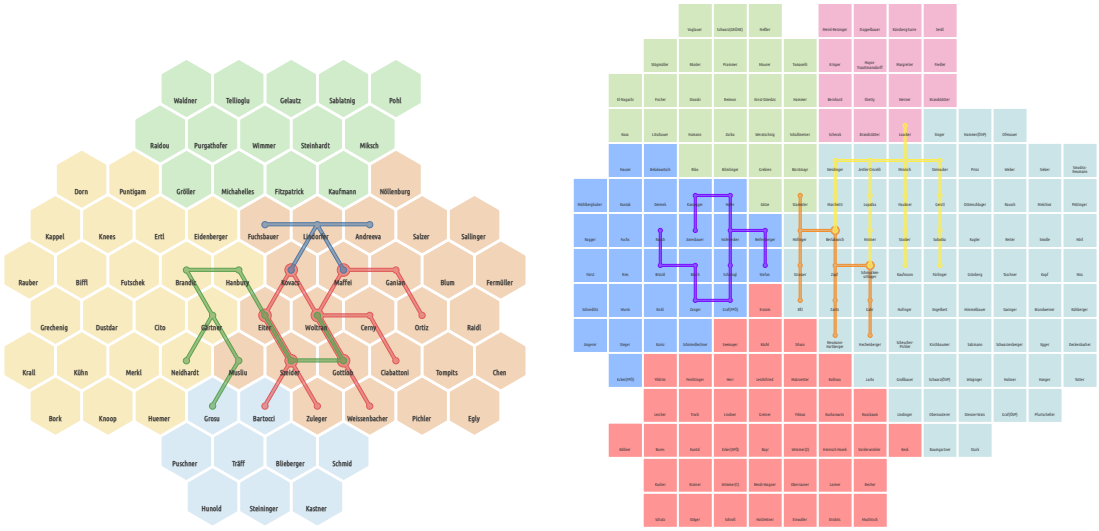
we repeated an updated version of the interview with the two experts from the preliminary interview and an additional expert. The provided manuscript is the one we used for the second interview phase, but for completeness we also added the questions on the rendering from the first phase.



(a) Parliament with hexagonal grid

(b) Parliament with square grid

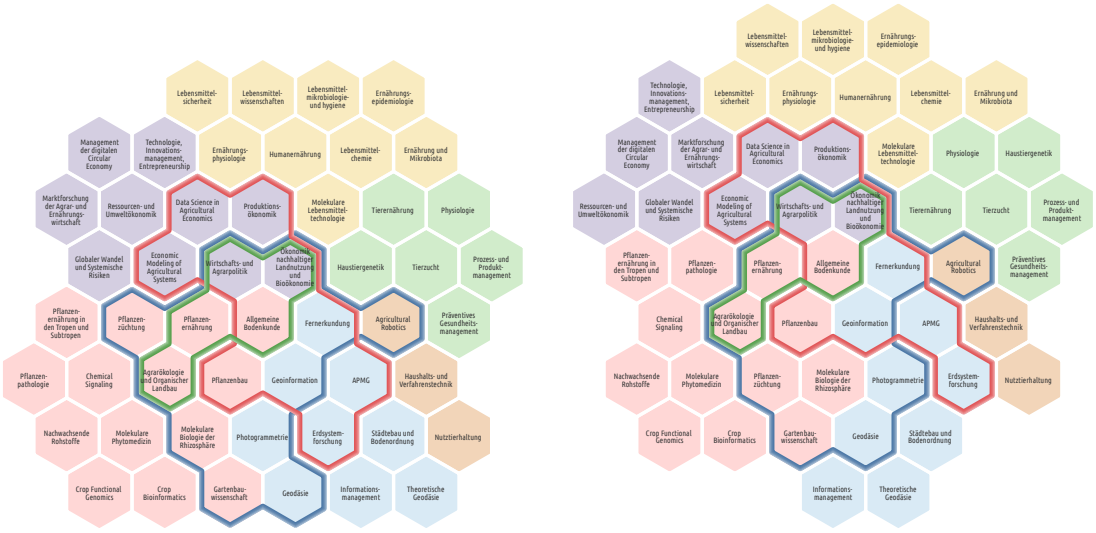
Figure A.1: Parliament with eight overlay sets visualized using MSE and relaxing the contiguity constraint as described in Section 5.4.3. The solution shown in (a) has a $PP_{C_2} = 0.477$ and a computing time of 2.8 s. Three of the eight overlay sets are not contiguous (dark green, purple and blue). The solution shown in (b) has a $PP_{C_2} = 0.471$ and a computing time of 2.2 s. Five of the eight overlay sets are not contiguous (dark green, blue, red, orange and gray). Parliament uses a color scheme that is a lighter variant of the typical political party colors used in Austria.



(a) Vienna

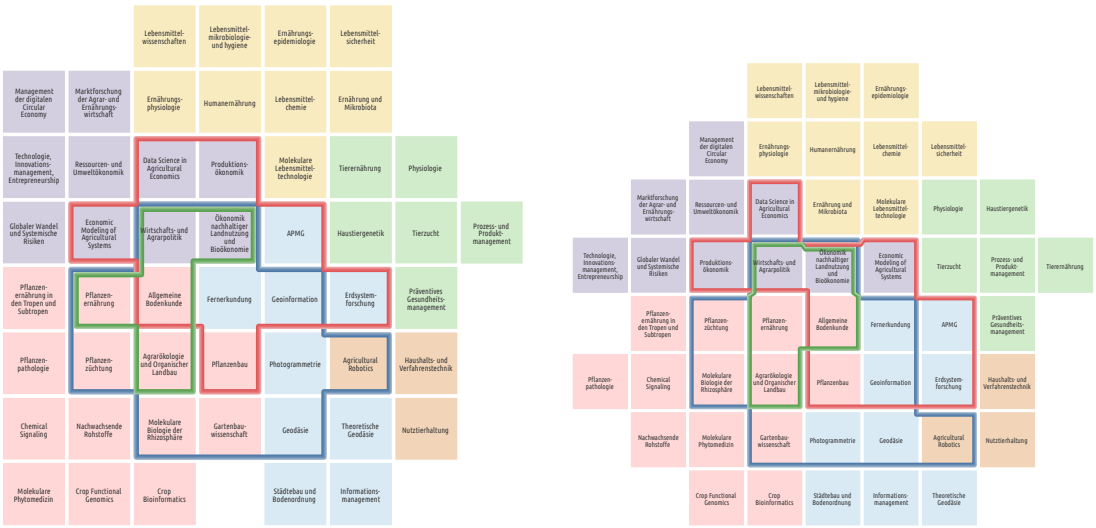
(b) Parliament

Figure A.2: Vienna (a) and Parliament (b) with three overlay sets visualized with the Kelp style. The corresponding visualizations with boundary style are given in Figure A.6a and Figure A.8a, respectively. See also Figure A.6a and Figure A.8a for compactness scores and running time. Parliament uses a color scheme that is a lighter variant of the typical political party colors used in Austria.



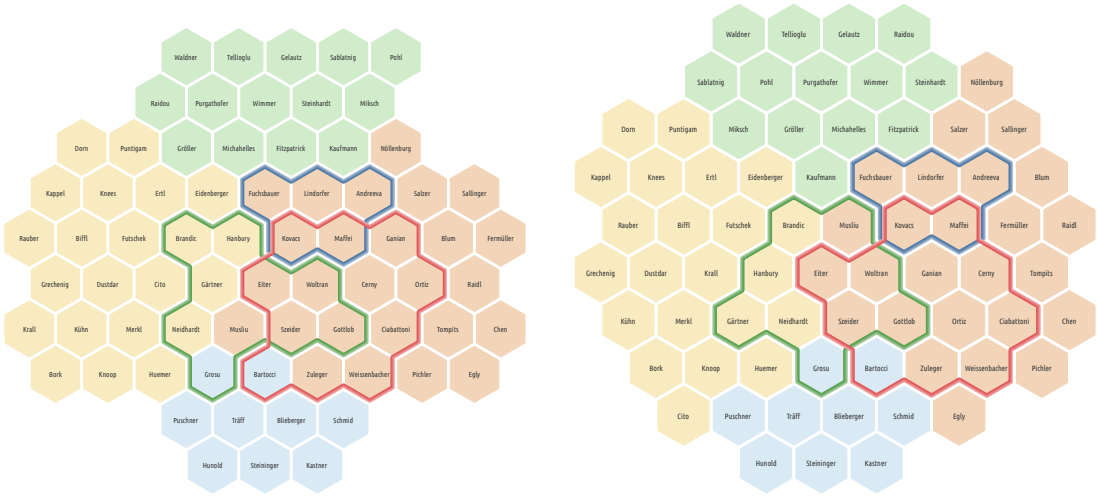
(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

Figure A.3: MosaicSets visualization for Bonn with three overlay sets and a hexagonal grid. (a) MSE with $PP_{C_2} = 0.521$ and running time of 0.7 s. (b) MSEA with $PP_{C_2} = 0.491$ and running time 0.7 s.



(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

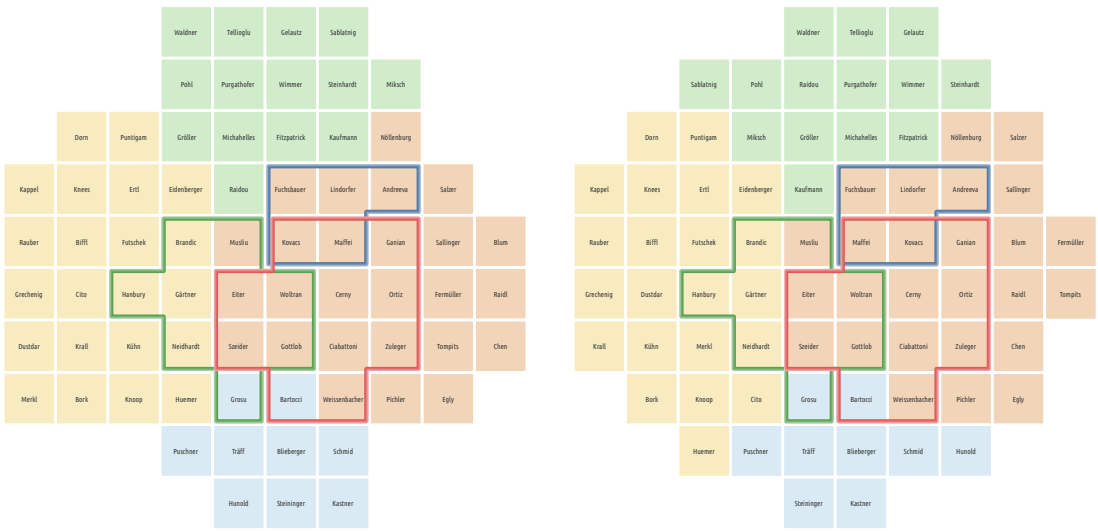
Figure A.4: MosaicSets visualization for Bonn with three overlay sets and a square grid. (a) MSE with $PP_{C_2} = 0.522$ and running time of 0.6 s. (b) MSEA with $PP_{C_2} = 0.509$ and running time of 0.8 s.



(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

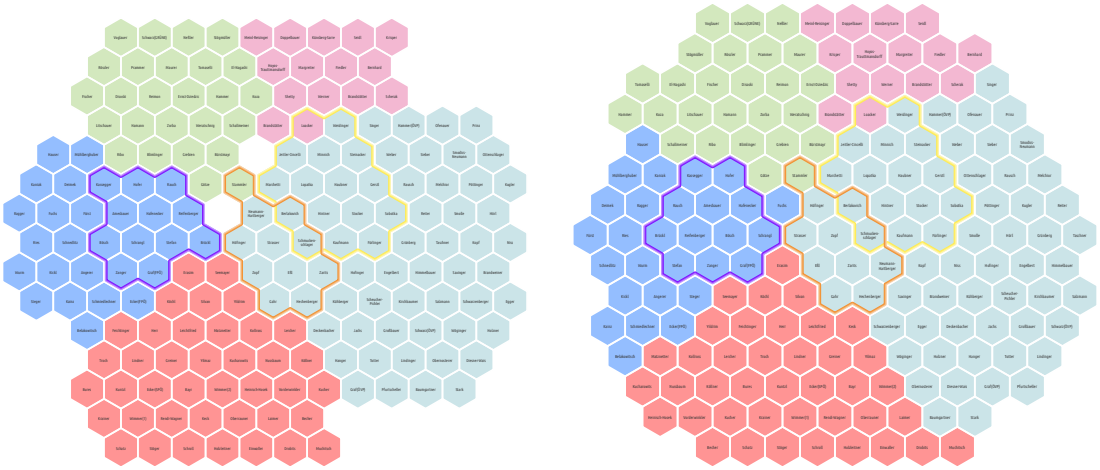
Figure A.5: MosaicSets visualization for VIENNA with three overlay sets and a square grid. (a) MSE with $PP_{C_2} = 0.574$ and running time of 0.7 s. (b) MSEA with $PP_{C_2} = 0.576$ and running time of 0.7 s

.



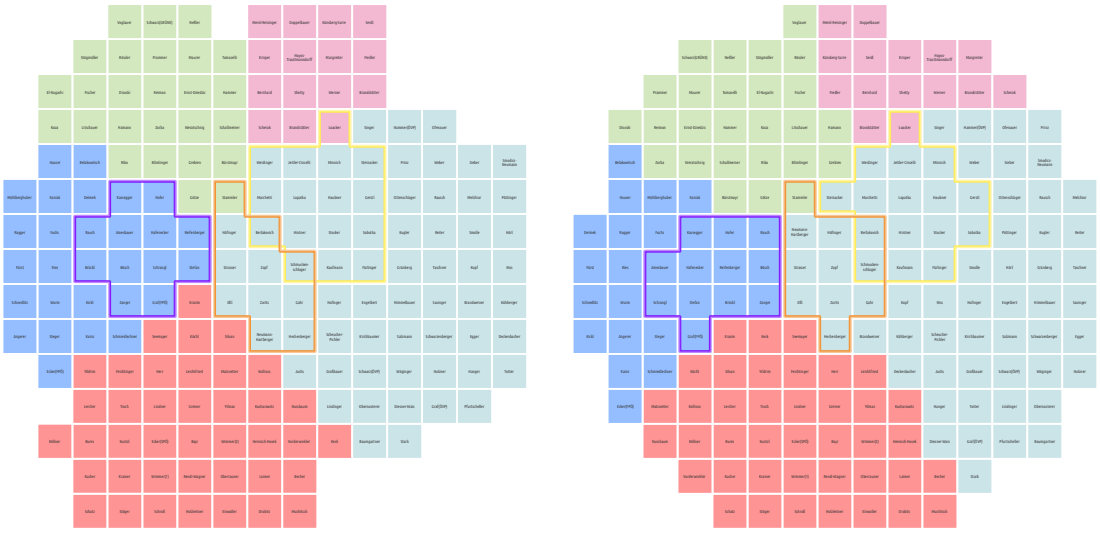
(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

Figure A.6: VIENNA with three overlay sets and square grid visualized with MosaicSets. (a) is MSE with $PP_{C_2} = 0.584$ and running time of 0.9 s. (b) MSEA with $PP_{C_2} = 0.532$ and a running time of 0.8 s.



(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

Figure A.7: PARLIAMENT with three overlay sets and hexagonal grid visualized with MosaicSets. (a) is MSE with $PP_{C_2} = 0.563$ and running time of 1.6 s. (b) MSEA with $PP_{C_2} = 0.543$ and a running time of 1.8 s. PARLIAMENT uses a color scheme that is a lighter variant of the typical political party colors used in Austria.



(a) eccentricity-based compactness (MSE) (b) as (a) but area fixed after first iteration (MSEA)

Figure A.8: PARLIAMENT with three overlay sets and square grid visualized with MosaicSets. (a) is MSE with $PP_{C_2} = 0.579$ and running time of 1.6 s. (b) MSEA with $PP_{C_2} = 0.512$ and a running time of 1.8 s. PARLIAMENT uses a color scheme that is a lighter variant of the typical political party colors used in Austria.

#	**Task**	MosaicSets Expert 1	MosaicSets Expert 2	MosaicSets Expert 3	Euler Diagrams	Frequency Grids
A1	Find/Select elements that belong to a specific set	●	●	●	●	●
A2	Find sets containing a specific element.	●	●	○	●	●
A3	Find/Select elements based on their set memberships	●	●	●	●	○
A4	Find/Select elements in a set with a specific set member-ship degree	○	●	●	-	○
A5	Filter out elements based on their set memberships.	-	-	-	-	○
A6	Filter out elements based on their set membership degrees	-	-	-	-	○
A7	Create a new set that contains certain elements.	-	-	-	○	○
B1	Find out the number of sets in the set family.	●	●	●	○	○
B2	Analyze inclusion relations.	●	●	●	●	●
B3	Analyze inclusion hierarchies	●	●	●	●	-
B4	Analyze exclusion relation	●	●	●	●	-
B5	Analyze intersection relation	●	●	●	●	●
B6	Identify intersections between k sets	●	●	●	○	●
B7	Identify the sets involved in a certain intersection	●	●	●	●	●
B8	Identify set intersections belonging to a specific set	●	●	●	●	●
B9	Identify the set with the largest / smallest number of pair-wise set intersections	●	●	●	○	○
B10	Analyze and compare set- and intersection cardinalities	●	●	●	●	●
B11	Analyze and compare set similarities	-	-	-	○	-
B12	Analyze and compare set exclusiveness	●	●	●	○	●
B13	Highlight specific sets, subsets, or set relations	-	-	-	n/a	●
B14	Create a new set using set-theoretic operation	-	-	-	○	-
C1	Find out the attribute values of a certain element	-	-	-	○	↖
C2	Find out the distribution of an attribute in a certain set or subset	-	-	-	○	-
C3	Compare the attribute values between two sets or subsets	-	-	-	○	-
C4	Analyze the set memberships for elements having certain attribute values	-	-	-	-	-
C5	Create a new set out of elements that have certain attribute values	-	-	-	-	-

Table A.1: Task taxonomy by Alsallakh et al. (2016) and assessment of MosaicSets, Euler diagrams and frequency grids (Micallef et al., 2012). Tasks can be classified as: (●) supported; (○) partially supported; (-) unsupported; or (↖) requires interactivity. For MosaicSets the classification is based on the opinion of the expert interviews. For Euler diagrams and frequency grids the classification is according to Alsallakh et al. (2016).

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

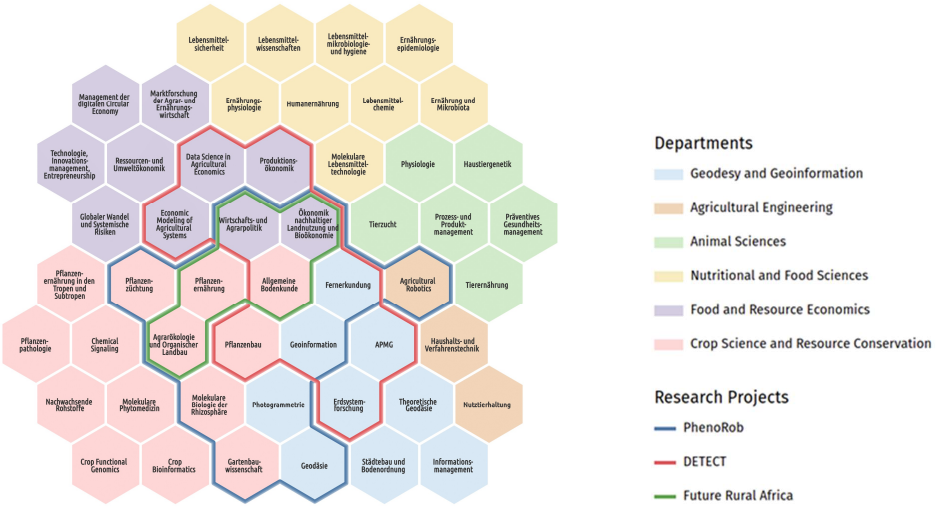
Expert Study

Scenario

The board of an institution (e.g., a university faculty) would like to improve its strategic planning processes by supporting discussions in meetings with informative visualizations. In particular, the board would like to visualize the division of its research groups into organizational units (e.g. departments) as well as important intra-institutional collaborations (e.g., major research projects involving several research groups from different departments). It is important that each research group is represented equally. Interactive components can also be used to represent the organizational structures on a web presentation to the public.

Introduction to our Visualization

In our approach, each research group is represented as a cell on a hexagonal or rectilinear grid. We enforce that all cells of the same department or of a project form a contiguous region. We illustrate the departments by coloring all cells corresponding to one department with the same color. Furthermore, each project is visualized by its region's boundary. We assign a distinct color to each project.



For the web presentation each legend entry is enriched such that a user can select and highlight each department individually. The projects are not displayed by default, but can be added to the visualization manually.

➔ Show SVG.

1

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Rechtwinkliges vs. sechseckiges Raster

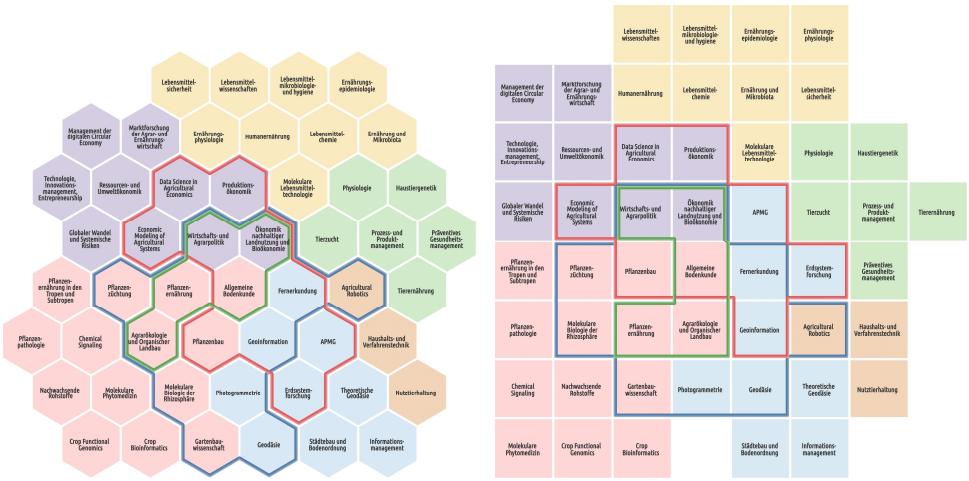


Figure: Hexagonal and Rectilinear Grid

- Which one is visually more appealing?
- Which one is more clear?
- What advantages and disadvantages do you see?

2

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Comparison of different rendering styles for the projects

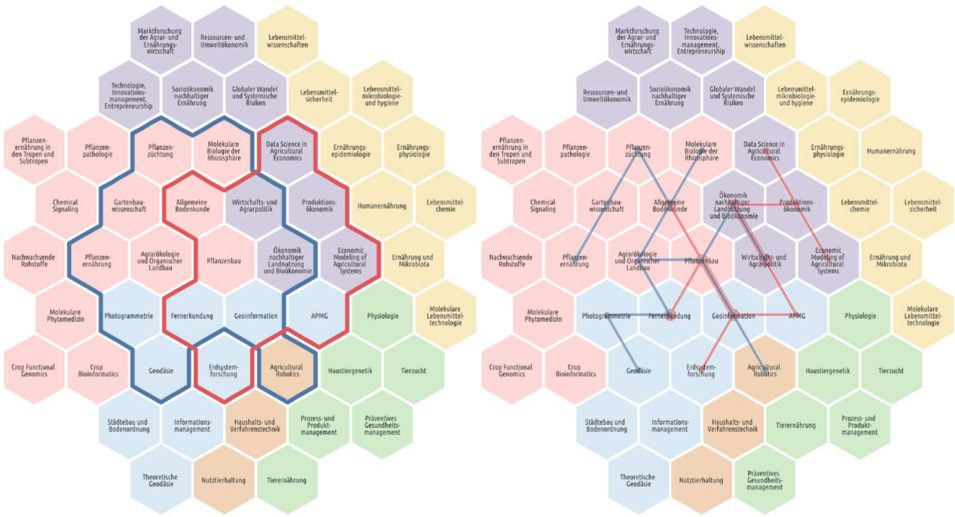


Figure 1: Kelp-Style

- Which one is visually more appealing?
- Which one is more clear?
- What advantages and disadvantages do you see?

3

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Compactness (MSE vs. MSEA)

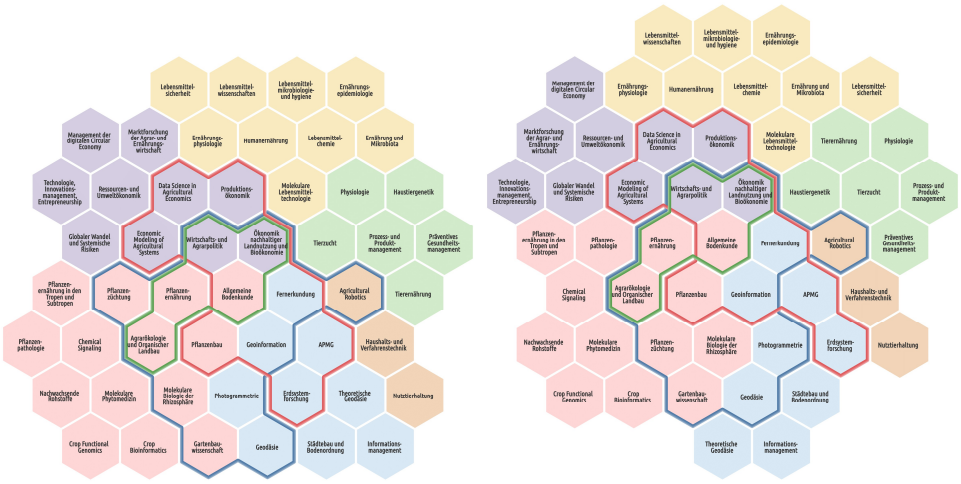


Figure 2: Left: MSE, Right: MSEA

- Welche Darstellung ist visuell ansprechender?
- Welche Darstellung ist übersichtlicher?
- Welche Vor- und Nachteile besitzen die Varianten?

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022

Page [3] from expert study performed in March 2022

Number of projects for static use-case.

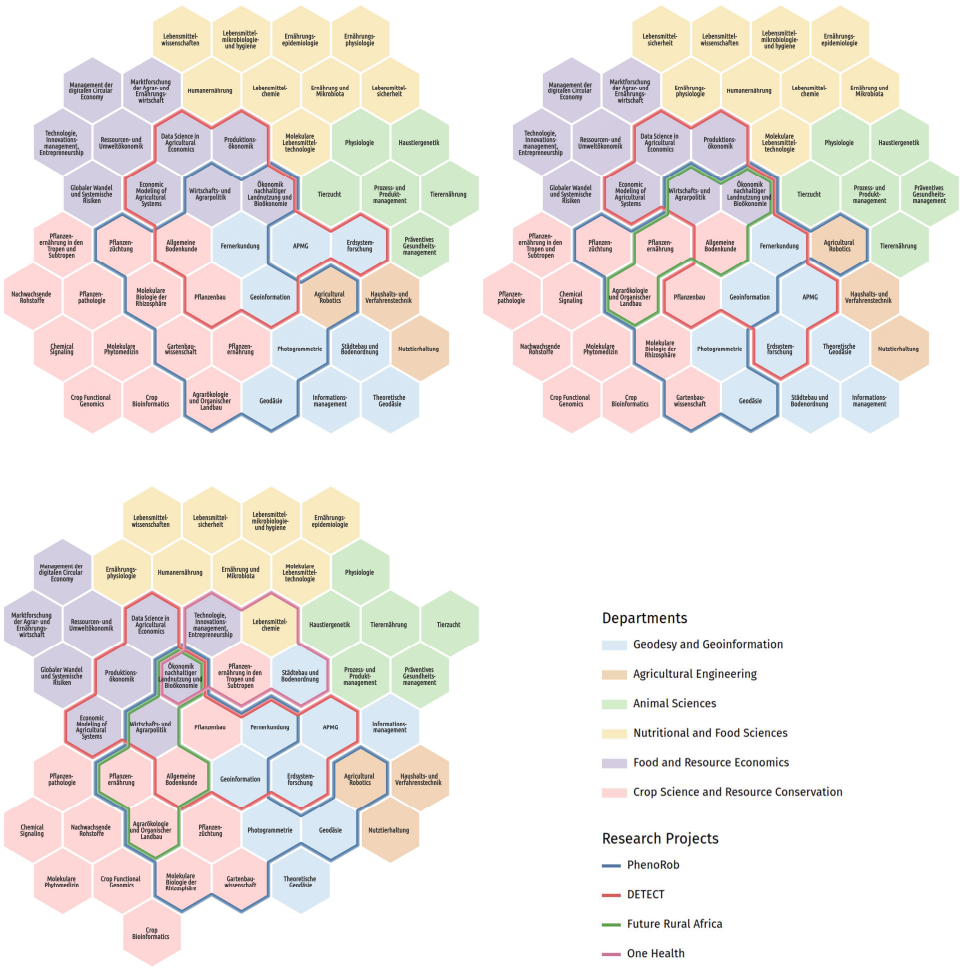


Figure 3: Visualization with 2, 3 and 4 research projects.

- The three figures differ in terms of the number of research projects represented. In your opinion, at what number of projects is the limit of what can be clearly presented with our approach?

5

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Static vs. Interactive.

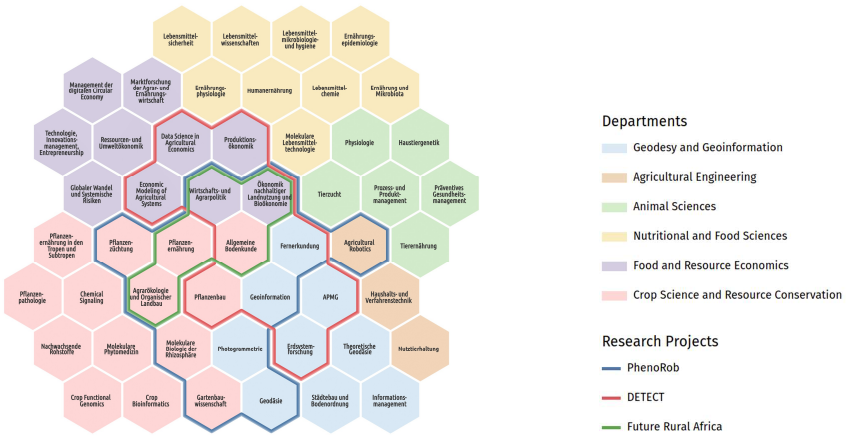


Figure 4: Static Visualization

- Can the interactive application improve clarity?
- Do you see any other advantages or disadvantages with the interactive version?

6

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Comparison to a manually generated visualization by an expert.

In the following, we compare a visualization created manually by an expert against a visualization generated by our approach. Please consider only the layout when answering the following questions and do not refer to design elements such as font type and size or color choice.

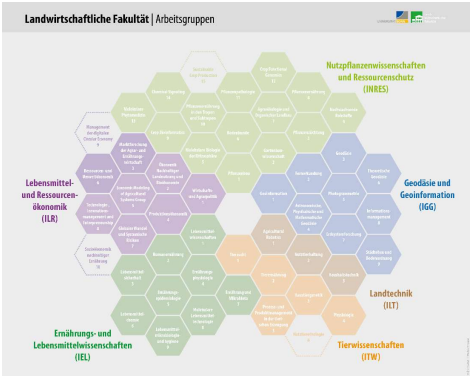


Figure 5: Manually Generated

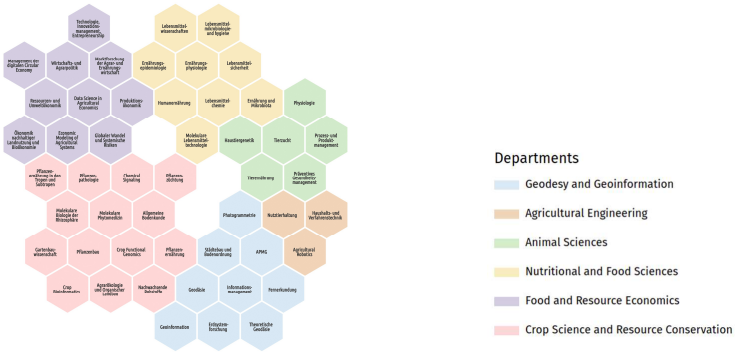


Figure 6: Our Solution

- Which layout is visually more appealing?
- Which one is more clear?
- What advantages and disadvantages do you see?

Only to the expert that designed the visualization manually:

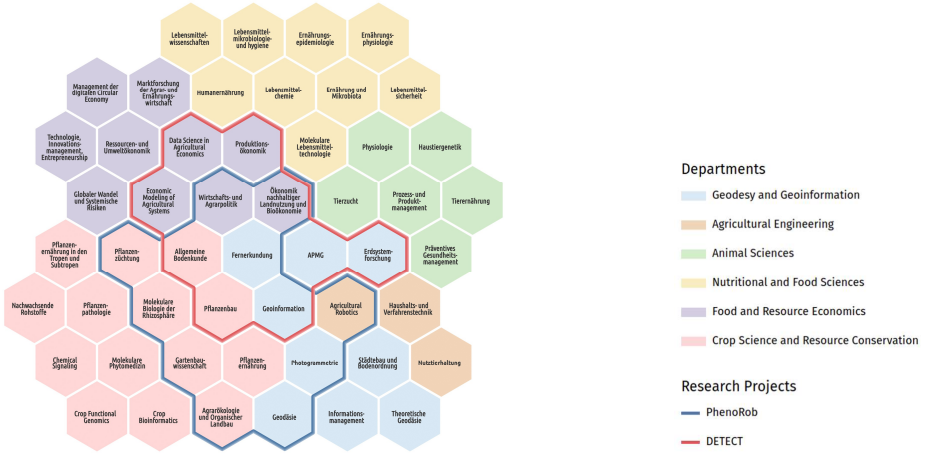
- What criteria did you use to create the visualization?
- Are these criteria also sufficiently considered in our visualization?

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

Tasks from State-Of-The-Art Paper

When designing a set visualization, it is important to determine which tasks should be supported. To this end, Alsallakh et al. gives a list of general tasks that are supported by existing visualization techniques. In the following, we ask you to classify our visualization with respect to these tasks. For each task you can choose between 'fully supported', 'partially supported' and 'not supported'.



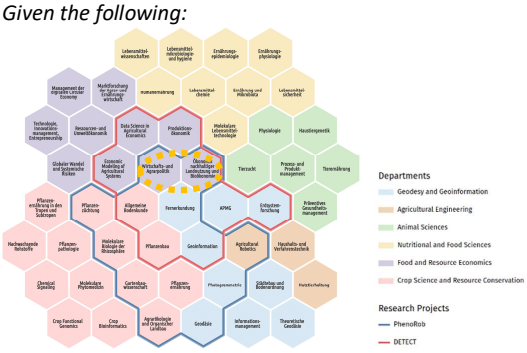
	Task	Correct answer?	Task supported?		
			Yes	Part.	o
A1	*Find all research groups that belong to PhenoRob.*				
A2	*Find the department and projects in which the research group 'Photogrammetrie' is contained.*				
A3	*Find all research groups that are part of the IGG but not of PhenoRob.*				
A4	*Find all research groups that are part of one department and two projects.*				
A5	*Find all research groups that are not part of both IGG and PhenoRob.*				x
A6	*Find all research groups that are not part of two projects.*				x
A7	*Create a new set that contains certain elements.*				x
B1	*How many departments and projects exist?*				
B2	*Is PhenoRob included in IGG?*				
B3	*Is PhenoRob included in DETECT, and DETECT in turn is included in IGG?*				
B4	*Do PhenoRob and IEL NOT have a joint research group?*				
B5	*Is there a joint research group of PhenoRob and IGG?*				
B6	*Find the intersection of PhenoRob and IGG.*				

8

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022

Page [3] from expert study performed in March 2022

B7	*Given the following:*  *Find the department and projects involved in the intersection (orange).*				
B8	*Find all projects in which the IGG is involved.*				
B9	*Which department participates in the least/ most projects?*				
B10	• *Estimate the number of research groups that are part in PhenoRob and IGG.* • *Compare the number of research groups of ILR and IEL.*				
B11	*Analyze and compare set similarities.*				x
B12	*Compare the number of IGG research groups not involved in any project to the number of IEL research groups not involved in any project.*				
B13	*Highlight sets, subsets or set relations.*				x
B14	*Create a new set using set-theoretic operations.*				x

9

Expert Study MosaicSets **2022**

Pages [1-2,4-10] from expert study performed in June 2022
Page [3] from expert study performed in March 2022

General Questions

(1) Can you imagine using our visualization for future representations? If this is not the case, what would be necessary?
(2) Where might such visualizations be used within the faculty?
(3) On which occasions (new appointments, new research projects, etc.) would you recreate a visualization? How often do these cases usually occur?
(4) How long may the generation of such a visualization take? For example, is a time of less than one minute sufficient?

Only to the expert that designed the visualization manually:

(1) Approximately how long did it take you to create the visualization?

(2) What limitations do you see?
    a. What additions would you like to see in the visualization?
    b. Regarding the dynamic visualization, what additional interaction options would you like to see?
(3) Is there any further feedback you want to give?

10

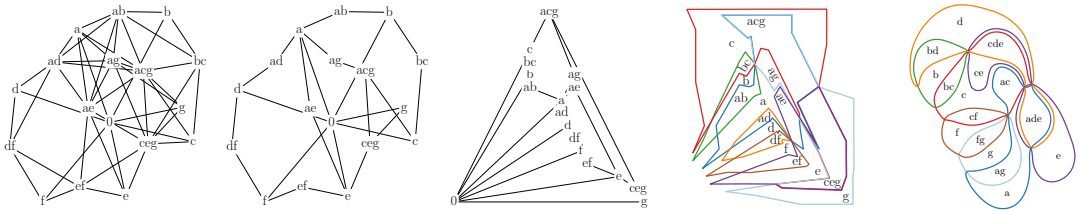
Appendix B

Supplemental Euler Diagrams

B.1 Euler Diagram Simplification

In this section, we provide supplemental material that is not included in the main manuscript due to the readability of the document. First, we provide an additional visualization of the workflow. Afterwards, we show the influence of several parameter settings on the resulting Euler diagrams.

Figure B.1 show the complete workflow for creating a smoothed Euler diagram from a superdual graph. In this example, the set system of the director Art Camacho from MovieDB is displayed.



(a) superdual graph (b) planar support (c) planar embedding (d) Euler diagram (e) resulting diagram

Figure B.1: Euler diagram of the set system corresponding to the director Art Camacho from MovieDB.

B.2 Parameter Influence

The problem definition MCSetSystemSimplification contains an objective function with a weighted sum of three individual objective functions. We show a single example of the simplification capabilities of our method in Figure 6.5. In this document, we show additional examples using different parameter values in this supplemental material.

Figure B.2 shows eight simplification steps of the set system of Leigh Slawner.

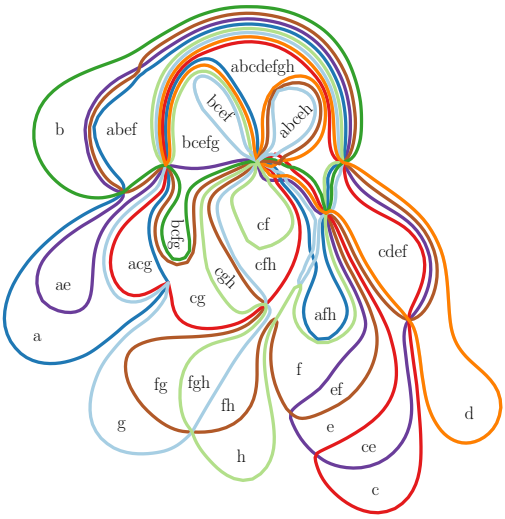
The total number of concurrencies of the Euler diagram with $\alpha = 0.01$ (Figure B.2a) is 23, which decreases to 2 for $\alpha = 3.75$ (Figure B.2h).

Figure B.3 shows four simplification steps of the set system of Joel Schoenbach. The total number of concurrencies of the Euler diagram with $\alpha = 0.01$ (Figure B.3a) is 16, which decreases to 0 for $\alpha = 2.0$ (Figure B.3d).

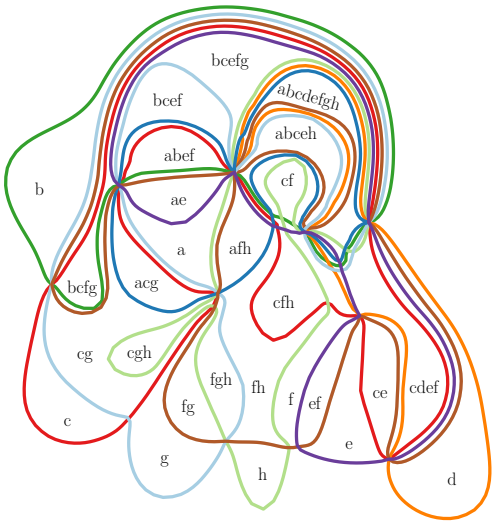
Figure B.4 shows three simplification steps of the set system of Jim Wynorski. The total number of concurrencies of the Euler diagram with $\alpha = 0.01$ (Figure B.4a) is 7, which decreases to 0 for $\alpha = 1.0$ (Figure B.4c).

Figure B.5 shows three simplification steps of the set system of Art Camacho. The total number of concurrencies of the Euler diagram with $\alpha = 0.01$ (Figure B.5a) is 4, which decreases to 0 for $\alpha = 1.5$ (Figure B.5c).

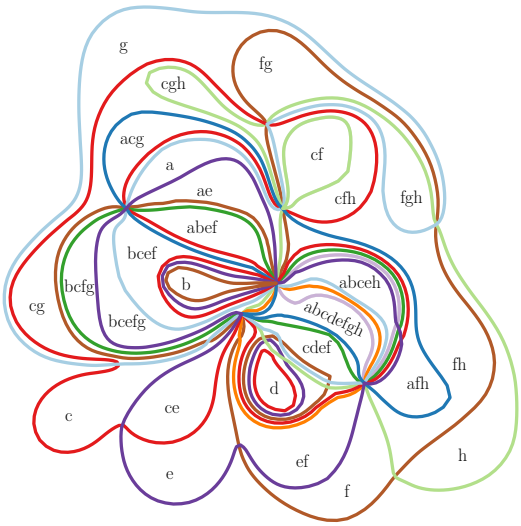
Figure B.6 shows three solutions of the set system of Leigh Slawner using different values for β. The parameter α is set to 2.25 for all three examples. In the left Euler diagram with $\beta = 0.1$, the number of faces adjacent to the outer face is 5, which is increased to 8 faces for $\beta = 1.0$. The Euler diagram shown in Figure B.6a evaluates to $f_{\text{weight}} = 92$, $f_{\text{concur}} = 7$ and $f_{\text{outer}} = 5$. The diagram shown in Figure B.6c evaluated to $f_{\text{weight}} = 90$, $f_{\text{concur}} = 7$ and $f_{\text{outer}} = 8$.



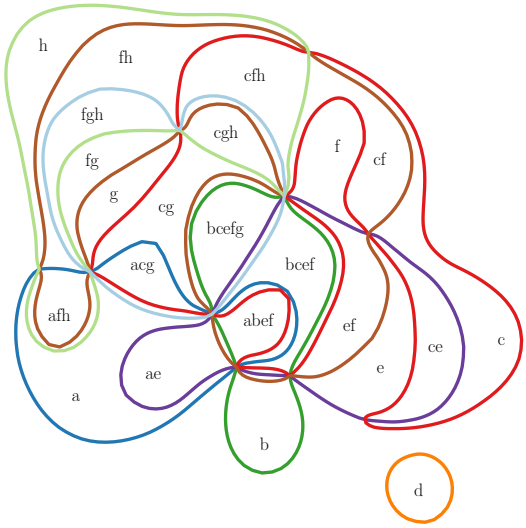
(a) $\alpha = 0.01$



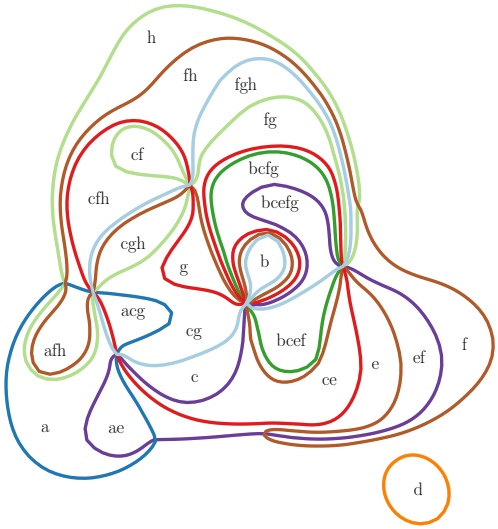
(b) $\alpha = 0.1$



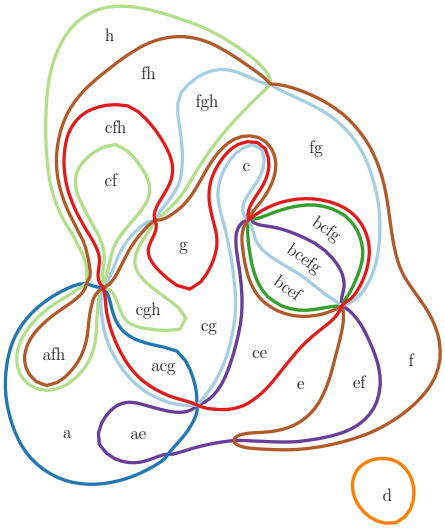
(c) $\alpha = 0.25$



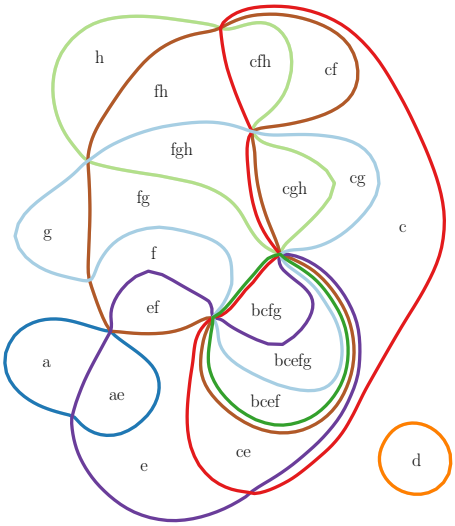
(d) $\alpha = 1.5$



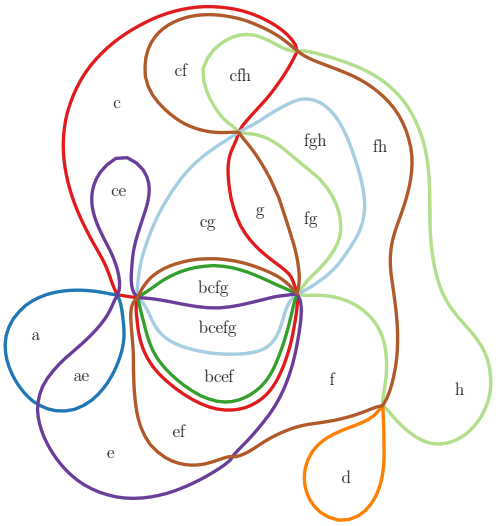
(e) $\alpha = 2.25$



(f) $\alpha = 2.75$

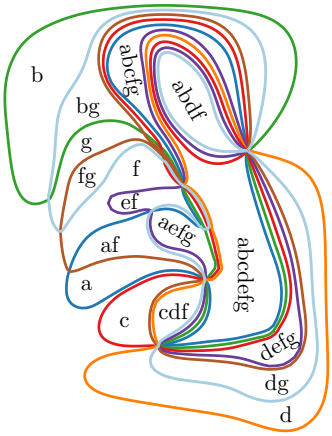


(g) $\alpha = 3.0$

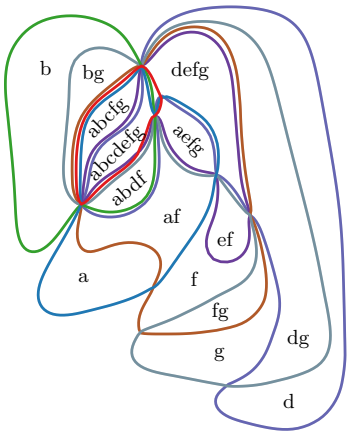


(h) $\alpha = 3.25$

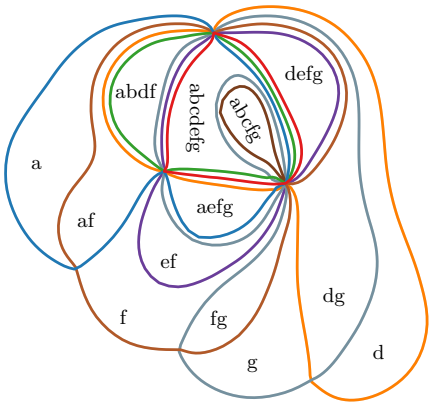
Figure B.2: Influence of parameter α on the set system corresponding to the director Leigh Slawner from MovieDB. Parameter β is fixed to 0.1 for all results.



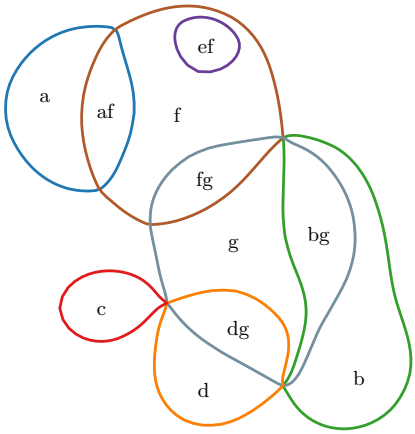
(a) $\alpha = 0.01$



(b) $\alpha = 1.25$

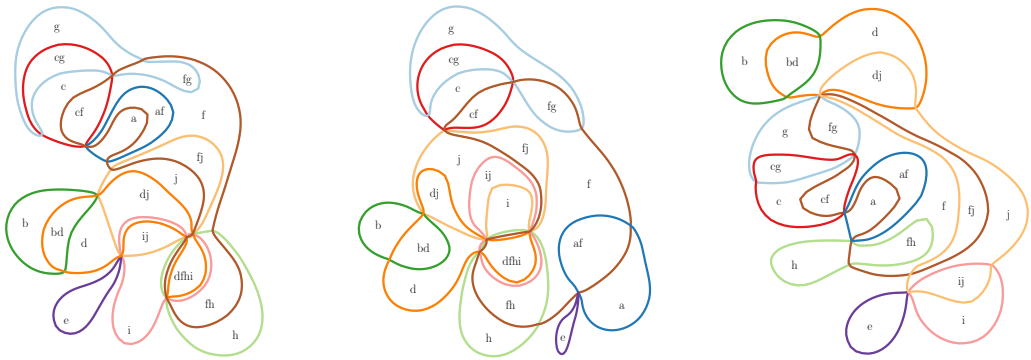


(c) $\alpha = 1.75$



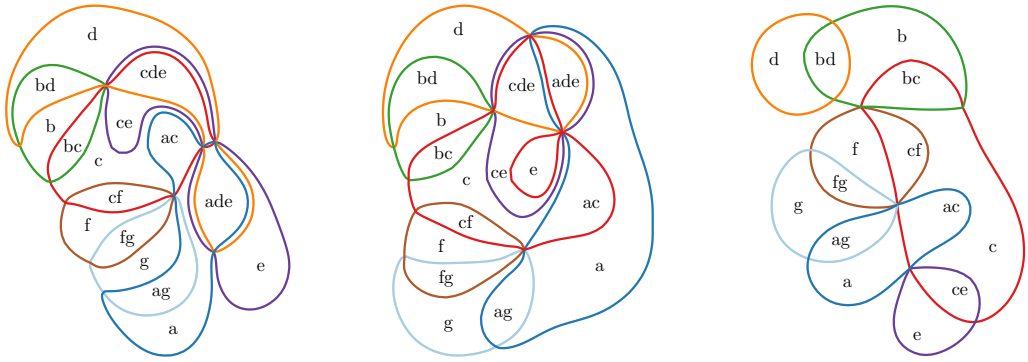
(d) $\alpha = 2.0$

Figure B.3: Influence of parameter α on the set system corresponding to the director Joel Schoenbach from MovieDB. Parameter β is fixed to 0.1 for all results.



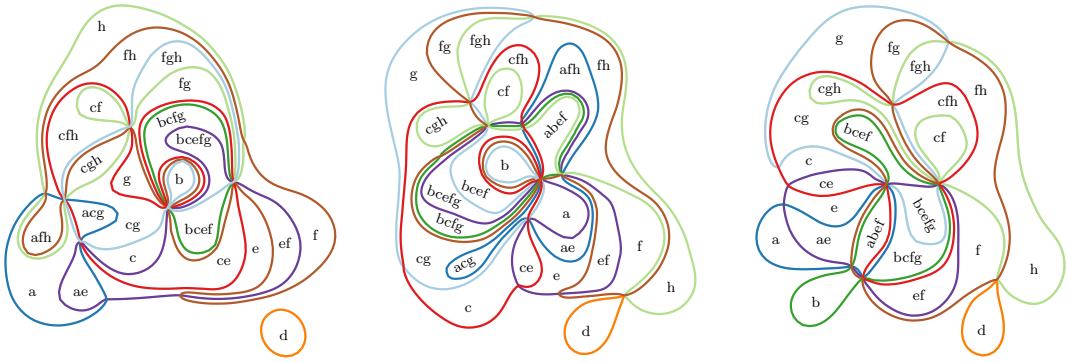
(a) $\alpha = 0.01$ (b) $\alpha = 0.2$ (c) $\alpha = 1.0$

Figure B.4: Influence of parameter α on the set system corresponding to the director Jim Wynorski from MovieDB. Parameter β is fixed to 0.1 for all results.



(a) $\alpha = 0.01$ (b) $\alpha = 0.15$ (c) $\alpha = 1.5$

Figure B.5: Influence of parameter α on the set system corresponding to the director Art Camacho from MovieDB. Parameter β is fixed to 0.1 for all results.



(a) $\beta = 0.1$ (b) $\beta = 0.75$ (c) $\beta = 1.0$

Figure B.6: Influence of parameter β on the set system corresponding to the director Leigh Slawner from MovieDB. Parameter α is fixed to 2.25 for all results.