Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 862

Johannes Oehrlein

Exact Optimization Algorithms for the Aggregation of Spatial Data

München 2020

Verlag der Bayerischen Akademie der Wissenschaften

ISSN 0065-5325 | ISBN 978-3-7696-5274-1

Diese Arbeit ist gleichzeitig als elektronische Dissertation
bei der Universitäts- und Landesbibliothek Bonn veröffentlicht:
https://nbn-resolving.org/urn:nbn:de:hbz:5-60713

Veröffentlichungen der DGK

Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften

Reihe C | Dissertationen | Heft Nr. 862

Exact Optimization Algorithms for the Aggregation of Spatial Data

Von der Landwirtschaftlichen Fakultät

der Rheinischen Friedrich-Wilhelms-Universität Bonn

zur Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation

Vorgelegt von

Johannes Oehrlein

Geboren in Schweinfurt

München 2020

Verlag der Bayerischen Akademie der Wissenschaften

ISSN 0065-5325 ISBN 978-3-7696-5274-1

Diese Arbeit ist gleichzeitig als elektronische Dissertation
bei der Universitäts- und Landesbibliothek Bonn veröffentlicht:
https://nbn-resolving.org/urn:nbn:de:hbz:5-60713

Adresse der DGK:



Ausschuss Geodäsie der Bayerischen Akademie der Wissenschaften (DGK)

Alfons-Goppel-Straße 11 ● D – 80 539 München

Telefon +49 – 331 – 288 1685 ● Telefax +49 – 331 – 288 1759

E-Mail post@dgk.badw.de ● http://www.dgk.badw.de

Prüfungskommission:

Vorsitzender:	Prof. Dr.-Ing. Theo Kötter
Referent:	Prof. Dr.-Ing. Jan-Henrik Haunert
Korreferenten:	Prof. Dr. Anne Driemel Prof. Dr.-Ing. Martin Kada
Fachnahes Mitglied:	Prof. Dipl.-Ing. Dr. techn. Wolf-Dieter Schuh

Tag der mündlichen Prüfung: 01.07.2020

© 2020 Bayerische Akademie der Wissenschaften, München

Alle Rechte vorbehalten. Ohne Genehmigung der Herausgeber ist es auch nicht gestattet,
die Veröffentlichung oder Teile daraus auf photomechanischem Wege (Photokopie, Mikrokopie) zu vervielfältigen

ISSN 0065-5325 ISBN 978-3-7696-5274-1

Danksagung

Die Arbeit der vergangenen Jahre findet in dieser Veröffentlichung ihren Abschluss. Der Weg hierher war nicht immer ein leichter, doch stets spannend und von interessanten Erfahrungen geprägt. An dieser Stelle möchte ich mich bei den Menschen bedanken, ohne die dieser Weg um einiges beschwerlicher, wenn nicht gar unpassierbar gewesen wäre.
An erster Stelle möchte ich mich aufrichtig bei meinem Betreuer Jan-Henrik Haunert bedanken. Die Geoinformatik als Forschungsbereich habe ich als Student in Würzburg in einer von ihm geleiteten Veranstaltung kennengelernt, bei der er gleich mein Interesse wecken konnte. Vermutlich haben dazu auch sein Enthusiasmus und seine Akribie hinsichtlich der Bearbeitung von Problemen der Geoinformatik ihren Teil beigetragen, mit denen er mich auch später motivieren konnte; sei es bei der Behandlung neuer Probleme oder der genaueren Untersuchung bereits betrachteter. Vielen Dank für deine Unterstützung und dein Vertrauen!
Neben meinem Betreuer gilt mein Dank auch den beiden weiteren Gutachtern Anne Driemel und Martin Kada, die dieser Arbeit mit ihren hilfreichen Kommentaren den letzten Schliff gegeben haben.
Kern dieser Arbeit sind unter anderem von mir veröffentlichte Arbeiten. Daher gilt ein großer Dank all meinen Koautoren: für aufschlussreiche Diskussionen, interessante Ideen und natürlich die gemeinsam produzierten Ergebnisse. In erster Linie geht mein Dank hier an Jan-Henrik Haunert; aber auch die insbesondere von Youness Dehbi, Thomas van Dijk und Benjamin Niedermann mit mir geteilten Erfahrungen waren von großem Wert. Auch für die gemachten Erfahrungen bei der Erarbeitung weiterer, in dieser Arbeit nicht direkt berücksichtigter (zukünftiger) Veröffentlichungen bin ich allen nicht genannten Koautoren sehr dankbar.
Einen nicht zu vernachlässigenden Beitrag zum Gelingen dieser Arbeit haben all meine Kolleginnen und Kollegen geleistet. Ich schätze mich sehr glücklich, während meiner Promotion so viele Personen kennen gelernt zu haben, die mich nicht nur fachlich, sondern auch persönlich vorangebracht haben, sei es im Büro oder außerhalb. Auf die vergangenen Jahre blicke ich nicht nur mit Stolz auf das Geleistete, sondern auch mit viel Freude auf das Erlebte zurück; auf veranstalte Spieleabende, gemeinsame sportliche Aktivitäten, Film- und Kinoerlebnisse, diverse Ausflüge mit oder ohne antreibende Rätsel und vieles mehr.
Reich an Erfahrungen war die Zeit mit euch am Institut für Geoinformatik und Fernerkundung an der Universität Osnabrück. Vielen Dank für die zahlreichen schönen Erinnerungen und eure Unterstützung!
In der Arbeitsgruppe Geoinformation des Instituts für Geodäsie und Geoinformation an der Universität Bonn habe ich mich nicht minder wohlgefühlt und noch mal mehr Unterstützung erfahren. Ich danke euch herzlich für die vergangenen Jahre!
Namentlich hervorheben möchte ich hier gerne Benjamin Niedermann, dessen fachlichen Beitrag ich bereits (hoffentlich) ausreichend gewürdigt habe. Deine persönlichen Ratschläge aus deinem Promotionserfahrungsschatz oder einfach als Freund, waren und sind von großem Wert. Dankeschön!
Abschließend möchte ich meinen allergrößten Dank meiner Frau und unserer Familie, insbesondere meinen Eltern und Geschwistern, ausdrücken, die, zusammen, das Fundament für all das bilden, wofür ich mich hier bedanken konnte.

Kurzfassung

Die Aggregation räumlicher Daten ist ein verbreitetes Problem in der Geoinformatik. Dahinter verbirgt sich das Zusammenfassen von Objekten oder Funktionen zur Gewinnung einer weniger komplexen Repräsentation. Dazu gibt es verschiedene Anlässe: Weniger komplexe Daten ermöglichen oft eine einfachere Verarbeitung durch Algorithmen. Zudem erlauben sie eine vereinfachte Darstellung, wie sie im Bereich der Kartengeneralisierung Ziel ist.
In dieser Arbeit werden Probleme der Aggregation räumlicher Daten untersucht. Diese werden zunächst als Optimierungsprobleme formalisiert. Dazu wird jeweils eine Funktion definiert, die die Qualität gültiger Lösungen bewertet. Anschließend wird ein Algorithmus präsentiert, der stets eine Lösung mit bestmöglicher Bewertung findet. Diese Gütegarantie geht im Allgemeinen auf Kosten der Berechnungsdauer, was ein Grund für die weit verbreitete Anwendung von Heuristiken ist. Jedoch liegen die Vorteile einer optimalen Lösung auf der Hand: Manchmal ist eine „gute“ Lösung nicht ausreichend. Darüber hinaus können exakte Lösungen zum Vergleich herangezogen werden, um nicht-exakte Verfahren hinsichtlich ihrer Qualität zu bewerten. Dies ist besonders für Heuristiken interessant, deren Lösungsqualität nur empirisch ermittelbar ist. Eine weitere Stärke exakter Verfahren ist, dass sie zur Überprüfung zugrunde liegender Modelle herangezogen werden können.
In dieser Arbeit werden aus dieser Motivation heraus entstandene Aggregationsverfahren vorgestellt. Durch den räumlichen Charakter der untersuchten Daten spielen dabei neben semantischen auch geometrische Aspekte eine Rolle, wenn auch in unterschiedlichen Maßen.
Das erste vorgestellte Problem betrifft die Visualisierung von Straßennetzen in Navigationskarten. Bei gegebenem Standort wird eine übersichtliche Darstellung der Umgebung gesucht. Zu diesem Zweck wird eine Äquivalenzrelation auf möglichen Navigationszielen eingeführt, die die Grundlage der Aggregation bildet. Der vorgestellte Algorithmus aggregiert effizient die größtmögliche Anzahl äquivalenter Ziele.
Des Weiteren wird eine Klasse aus der Literatur bekannter Problemen behandelt, welche die Aggregation von Flächen in größere, zusammenhängende Regionen betreffen. Diese Probleme sind **NP**-vollständig, d. h. effiziente Algorithmen existieren vermutlich nicht. Es gelingt jedoch, bestehende exakte Verfahren um circa eine Größenordnung zu beschleunigen.
Ein weiteres betrachtetes Problem betrifft die Analyse der Verfügbarkeit von Grünflächen im urbanen Raum. Dazu werden, hypothetisch, mittels Flussnetzwerk Bewohner Grünflächen zugewiesen. Dadurch werden lokale Defizite sowie Muster in der Zugänglichkeit sichtbar.
Abschließend wird ein Mittel zum Erlernen von Präferenzen bei der Routenplanung vorgestellt. Basierend auf einer Auswahl an Trajektorien werden zwei mögliche Kriterien untersucht. Diese werden anschließend durch Linearkombination effizient zum bestmöglichen, ableitbaren Kriterium aggregiert.
Zusammenfassend werden in dieser Arbeit exakte Algorithmen als Antwort auf verschiedene Aggregationsprobleme in der Geoinformatik präsentiert. Insbesondere das betrachtete **NP**-vollständige Problem untermauert, wie erwartet, die Notwendigkeit heuristischer Verfahren. Diese sind gerade bei zeitkritischen Anwendungen von großer Bedeutung und insbesondere dank universell anwendbarer Metaheuristiken sehr beliebt. Die Ergebnisse dieser Arbeit sind jedoch ein weiterer Grund, bei der Suche nach Lösungsverfahren für Aggregationsprobleme mit exakten Verfahren zu beginnen. Die Gütegarantie spricht für sich. In einigen Fällen wurden sogar neue Algorithmen, die effizient und exakt sind, gefunden.

Abstract

The aggregation of spatial data is a recurring problem in geoinformation science. Aggregating data means subsuming multiple pieces of information into a less complex representation. It is pursued for various reasons, like having a less complex data structure to apply further processing algorithms or a simpler visual representation as targeted in map generalization.
In this thesis, we identify aggregation problems dealing with spatial data and formalize them as optimization problems. That means we set up a function that is capable of evaluating valid solutions to the considered problem, like a cost function for minimization problems. To each problem introduced, we present an algorithm that finds a valid solution that optimizes this objective function. In general, this superiority with respect to the quality of the solution comes at the cost of computation efficiency, a reason why non-exact approaches like heuristics are widely used for optimization. Nevertheless, the higher quality of solutions yielded by exact approaches is undoubtedly important. On the one hand, "good" solutions are sometimes not sufficient. On the other hand, exact approaches yield solutions that may be used as benchmarks for the evaluation of non-exact approaches. This kind of application is of particular interest since heuristic approaches, for example, give no guarantee on the quality of solutions found. Furthermore, algorithms that provide exact solutions to optimization problems reveal weak spots of underlying models. A result that does not satisfy the user cannot be excused with a mediocre performance of an applied heuristic.
With this motivation, we developed several exact approaches for aggregation problems, which we present in this thesis. Since we deal with spatial data, for all problems considered, the aggregation is based on both geometric and semantic aspects although the focus varies. The first problem we discuss is about visualizing a road network in the context of navigation. Given a fixed location in the network, we aim for a clear representation of the surroundings. For this purpose, we introduce an equivalence relation for destinations in the network based on which we perform the aggregation. We succeed in designing an efficient algorithm that aggregates as many equivalent destinations as possible.
Furthermore, we tackle a class of similar and frequently discussed problems concerning the aggregation of areal units into larger, connected regions. Since these problems are **NP**-complete, i.e. extraordinarily complex, we do not aim for an efficient exact algorithm (which is suspected not to exist) but present a strong improvement to existing exact approaches.
In another setup, we present an efficient algorithm for the analysis of urban green-space supply. Performing a hypothetical assignment of citizens to available green spaces, it detects local shortages and patterns in the accessibility of green space within a city.
Finally, we introduce and demonstrate a tool for detecting route preferences of cyclists based on a selection of given trajectories. Examining a set of criteria forming suitable candidates, we aggregate them efficiently to the best-fitting derivable criterion.
Overall, we present exact approaches to various aggregation problems. In particular, the **NP**-complete problem we deal with firmly underscores, as expected, the need for heuristic approaches. For applications asking for an immediate solution, it may be reasonable to apply a heuristic approach. This holds in particular due to easy and generally applicable metaheuristics being available. However, with this thesis, we argue for applying exact approaches if possible. The guaranteed superior quality of solutions speaks for itself. Besides, we give additional examples which show that exact approaches can be applied efficiently as well.

Contents

Danksagung iii

Kurzfassung iv

Abstract v

1 Introduction 1
- 1.1 Aggregation of spatial data . . . 2
- 1.2 Overview of existing aggregation approaches . . . 3
- 1.3 Optimization . . . 10
- 1.4 Goal and outline of this thesis . . . 12

2 Methodological background 15
- 2.1 Computational complexity theory . . . 15
- 2.2 Graph theory . . . 17
  - 2.2.1 Basic concepts . . . 18
  - 2.2.2 Graph algorithms . . . 20
  - 2.2.3 Flow networks . . . 28
- 2.3 Linear programming and (mixed-)integer linear programming . . . 36
  - 2.3.1 Linear programming . . . 36
  - 2.3.2 (Mixed-)Integer linear programming . . . 39

3 Location-dependent generalization of road networks based on equivalent destinations 49
- 3.1 Introduction . . . 50
- 3.2 Equivalent destinations in trees . . . 52
- 3.3 A linear-time algorithm for TreeSummary . . . 55
- 3.4 Map generalization . . . 56
- 3.5 Conclusion and Outlook . . . 63

4 A Cutting-Plane Method for Contiguity-Constrained Spatial Aggregation 65
- 4.1 Introduction . . . 66
- 4.2 A state-of-the-art model . . . 71
  - 4.2.1 A compact ILP without contiguity . . . 71
  - 4.2.2 Shirabe's model for contiguity-constrained spatial unit allocation . . . 72
  - 4.2.3 Area aggregation in map generalization . . . 73
- 4.3 Handling ILPs with large sets of constraints . . . 74
- 4.4 A new method for area aggregation using cutting planes . . . 77
  - 4.4.1 Constraints completing the ILP formulation . . . 78

4.4.2 Adding the constraints . . . 80
4.5 Results and discussion . . . 83
4.6 Conclusion . . . 89
4.A Appendix: Algorithms . . . 91

5 Analyzing the supply and detecting spatial patterns of urban green spaces via optimization 93
5.1 Introduction . . . 94
5.2 Related Work on Qualitative and Quantitative Analyses of Green Spaces . . . 97
5.3 Methodology . . . 99
5.3.1 Basic Model . . . 99
5.3.2 Analyzing the Resulting Clusters . . . 104
5.3.3 Deployment . . . 105
5.3.4 A running example . . . 108
5.4 Experiments and the methodology for the evaluation . . . 111
5.4.1 Data . . . 111
5.4.2 Setup . . . 112
5.4.3 Evaluation . . . 114
5.5 Conclusion . . . 121

6 Inferring routing preferences of bicyclists from sparse sets of trajectories 123
6.1 Introduction . . . 124
6.2 Related work . . . 126
6.3 Methodology . . . 128
6.3.1 Routing Model . . . 128
6.3.2 Classification of Trajectories . . . 129
6.3.3 Recognizing Unfavorable and Favorable Road Types . . . 130
6.3.4 Inferring Edge Weights . . . 130
6.4 Experiments . . . 131
6.4.1 Data . . . 131
6.4.2 Results of the Trajectory Classification . . . 132
6.4.3 Results of the Road-Type Classification . . . 133
6.4.4 Results of the Weight Inference . . . 135
6.5 Conclusion . . . 137
6.A Appendix: Aggregation of routing criteria . . . 140
6.A.1 Determining weighting factors corresponding to path optimality . . . 141
6.A.2 Segmenting a path into a minimum number of optimal subpaths . . . 147

7 Conclusion and outlook 151
7.1 Conclusion . . . 151
7.2 Outlook . . . 152

Bibliography 155

1 Introduction

The aggregation of spatial information is a fundamental process of map generalization and, thus, of map creation itself. Hence, there is a long history of aggregating spatial data. However, conditions have changed in the past century. In the early 20^{th} century at the latest, the formalization of this process started [RM07]. With the upcoming computer age, research interest in an automation of the map creation and, thus, generalization process increased. For generalization, according to Sarjakoski [Sar07], automation is the goal since the 1960s.

The technological progress in the last decades increased the availability and, hence, the usage of spatial data. An example of this trend is the phenomenon of volunteered geographic information (VGI), the contribution of large numbers of individuals, often amateurs, to the creation of geographic information. Goodchild [Goo07] attributes this effect to the increased availability of tools for acquiring geographic data. The availability of more and, partly, more complex data requires new means for processing. A possible reply is the development of faster, more sophisticated algorithms. A different approach is to decrease the complexity of the data, that is, to find a less complex, aggregated representation of the data.

With respect to cartography, another aspect is important concerning the aggregation of data. Besides the organization of the data, less complexity is also desired when it comes to visualizing the data. The role of aggregating spatial data, in particular with respect to map generalization, is examined in Section 1.1.

Due to its importance, there exist numerous publications on the topic of aggregation. In Section 1.2, we give an overview on existing approaches focusing on those that are of particular importance for aggregating spatial data. This includes algorithms that were designed for other applications originally, like image segmentation or statistical analysis.

In this thesis, we develop and describe various aggregation algorithms. Afterwards, we analyze each algorithm with respect to its running time and its correctness. Here, the correctness plays a particular important role since we decided to design our algorithms as exact optimization approaches. In Section 1.3, we give reasons for this decision. As a consequence of the aim to develop optimization algorithms, we contribute a formalization of some of the tackled problems as optimization problems if this has not happened before in the literature.

We conclude this chapter with an overview of the goals and the outline of this thesis in Section 1.4.

1.1 Aggregation of spatial data

Aggregating spatial data means subsuming multiple pieces of information into another one. This other piece of information can be a newly created object or a representative of the original, aggregated data. In any case, information is omitted in order to obtain a less complex representation. One reason why less complexity in the representation may be desired can be found in Chapter 2, Section 2.1, where we deal with the running time of algorithms. This running time depends on the complexity of the input data and, thus, a less complex representation can lead to a significantly improved running time. Another reason for reducing the complexity of data is to create a legible visualization. With respect to map generalization, this topic will be discussed in the following.

Aggregation in map generalization A *map* is an abstraction of the geographic reality and, thus, depicts only a subset of that reality [BW88, Sar07]. The term *generalization* describes the process of extracting this subset of important and general aspects of reality [BW88]. Depending on the exact process of designing a map, a varying number of generalization steps is involved. The International Cartographic Association (ICA) defines generalization (according to Sarjakoski [Sar07]) as follows:

> "The selection and simplified representation of detail appropriate to the scale and/or purpose of a map."

Hence, generalization is applied to both geometric and semantic information of geographic objects. According to Hake et al. [HGM02], cartographers differentiate between two major kinds of generalization: *object generalization* and *cartographic generalization*. The former is subdivided into *acquisition generalization* and *model generalization*. Acquisition generalization describes the process from the real world towards a model. Due to the real world's complexity and the resulting problem of describing it holistically, generalization takes place already in this first step of modeling. Model generalization, like acquisition generalization, is object oriented and differs mainly in the original data. In contrast to acquisition generalization, the original object, i.e., the input of the generalization process, is not the real world but already a model. Cartographic generalization, on the other hand, is more focused on the graphic representation of the geographic data. It deals, for example, with graphical restrictions on the representation of objects or with deriving new maps from existing ones.

Automated generalization is acknowledged to be a complex problem. Mackaness [Mac07] wonders "why a task so effortlessly performed by a human, has proved so hard to automate". He assumes different causes: The first explanation he gives underlines the subjectivity of the generalization process. Often, there are multiple different solutions which are "compromise[s] among a sometimes competing set of constraints". Furthermore, depending on the scale, a generalized map does not necessarily contain less information than the database but different yet related information. Here, Mackaness identifies another problem in developing generalization methods: Such tools need to abstract the same content of the original map in different ways, depending on the scale or purpose of the resulting map. In particular, he sees a problem in the evaluation of the result. Besides, according to Mackaness, generalization is not an operation that can be applied as the last step of map making since generalization based on geometries only is, often, doomed to fail. For the sake of a legible visualization,

generalizing geometries plays an important role. In general, however, the semantic context must not be forgotten. This is true also for aggregation processes. Hence, automated aggregation is based on databases providing sufficient information on the context rather than, for example, on a single map or visual representation.
While making progress in formalizing and automating generalization, various authors considered a splitting of the generalization process into several fundamental operators as reasonable. Regnauld and McMaster [RM07] give an overview of this development starting in 1942 with Wright [Wri42] identifying two major components, *simplification* and *amplification*. Over time, the number of fundamental operators increased in order to bring out the variety of operators more clearly. Hake et al. [HGM02], for example, list a total of seven fundamental operators, see Figure 1.1.

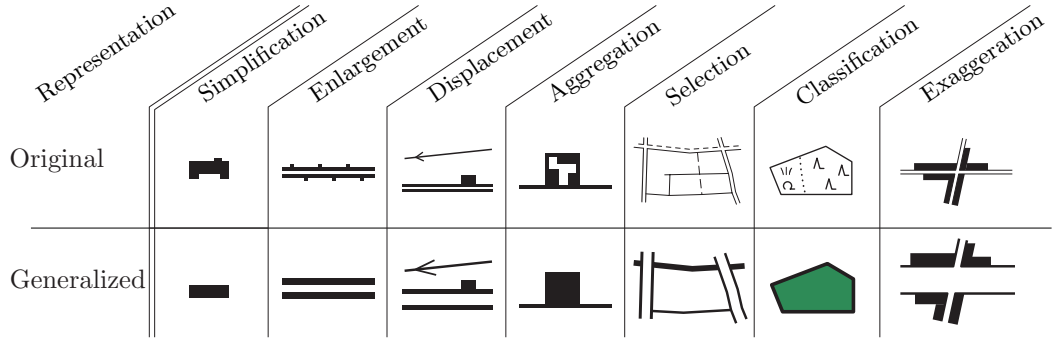


Figure 1.1: Fundamental generalization operators (according to Hake et al. [HGM02])

Hence, aggregation is one of the fundamental operators of map generalization [HGM02, RM07, MS92]. According to Hake et al. [HGM02], the operators listed in Figure 1.1 are applied for the sake of *semantic*, *geometric* and *temporal* generalization. In this thesis, we focus on semantic and geometric generalization. In Chapter 4, for example, we present an algorithm for aggregating areal units with respect to a semantic value. The algorithm aims for a user-specified compromise between similarity in attributes (semantic generalization) and compact shapes (geometric generalization) in the resulting aggregated objects. Here, the focus lies more on semantic generalization. In Chapter 3, on the other hand, we present an algorithm that is particularly useful for visualization and, thus, focuses more on geometric generalization. In that chapter, we aggregate destinations in a road network in order to select (see Figure 1.1) roads to be depicted in a map with a focus region. This can be a map describing how to get to a location or, as in our case, a map describing close features (and, in particular, roads) in more detail than distant features. As the semantic information on the roads, e.g. the road type, can be considered in this algorithm, again, a strict assignment to one kind of generalization, i.e. geometric or semantic, is not possible.

1.2 Overview of existing aggregation approaches

Since aggregation is a common problem in geoinformation science, there exists a multitude of algorithms dealing with it. In the following, we want to provide an overview of existing approaches. Due to the volume of scientific work on this matter, we limit ourselves to

approaches we consider important or particularly interesting with respect to processing geographic information. We start with rather generic algorithms for clustering problems for data with geometric information. The generic clustering algorithms presented here classically aim at point features. A reason why they are applicable to many problems is that higher-dimensional objects can also be considered as points in a feature space and, thus, be clustered with such algorithms. For example, there are algorithms [BDGK19, HLO12, ZHT06] that use distances between line features to define a metric space in which these line features are points.

The aggregation of point features is known to a wider public as clustering. This is mainly due to its application in the statistical analysis of data. Xu and Wunsch [XW05] as well as Jain [Jai10] provide an exhaustive overview on clustering algorithms. They present a variety of techniques and subsume clustering algorithms under the techniques used for their design. Fortunato [For10] terms the problem of clustering as *community detection in graphs* and reviews it extensively. In the following, we give an overview over a selection of these clustering algorithms. Each of the following algorithms aims at clustering some given set S. Initially, we follow Jain and distinguish mainly between two major groups of clustering algorithms, *hierarchical* and *partitional* ones.

Generic clustering problems and solutions Hierarchical algorithms aim for a dendrogram, a representation of a cluster hierarchy as a tree. The root of this tree represents the set S. Each node of the tree represents a subset $T \subseteq S$ and has two children representing non-empty sets forming a partition of T. This continues such that the leaves form the set S organized in singletons. The sought clustering is then a cross-section of this tree, see Figure 1.2. There exist multiple approaches for setting up the dendrogram which either build the tree in an agglomerative manner (i.e., starting at the singletons and continuing bottom-up) or divisive manner (i.e., top-down beginning at the root). In every construction step, the decision which sets to agglomerate or how to divide a given set into partitions is made based on a difference defined for the clusters. *Single linkage*, for example, considers the closest pair of points of two clusters for defining the distance between the clusters. It is applied (among others) by Mackaness and Mackechnie [MM99] in order to detect junctions in road networks. They want to focus on local accumulations of points rather than outliers. That is why they decided in favor of single linkage instead of, for example, *complete linkage*, which considers the maximum distance of points as decisive for the distance of the clusters.

In contrast to hierarchical clustering algorithms, partitional clustering algorithms find all clusters simultaneously [Jai10]. Among these, *k-means* is a very popular and simple approach [Jai10, XW05]. The term is used both for the most common algorithm, which has been designed in various fields independently, and the problem itself: Given a finite set S and a number k, find a partition of S into subsets S_i with $i = 1, \ldots, k$ with centers m_i such that $\sum_{i=1}^{k} \sum_{x \in S_i} ||x - m_i||^2$ is minimized, where $|| \cdot ||$ denotes the Euclidean distance. Efficient solutions to this problem exist only for the one-dimensional case [GLM+17], which plays an important role in cartography when it comes to specifying class intervals with natural breaks for choropleth maps [HGM02]. Due to the general problem's high complexity (more precisely, its **NP**-hardness [MNV09]), it is common to deal with it heuristically. Starting from an initial choice of k centers, Lloyd [Llo82] presented an algorithm that repeatedly assigns every element of S to the subset S_i with the closest center m_i and updates each

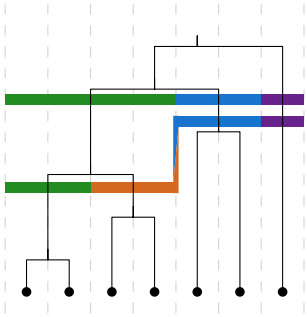


Figure 1.2: Dendrogram of a hierarchical cluster algorithm. Depending on the cross-section, three or four clusters are found.

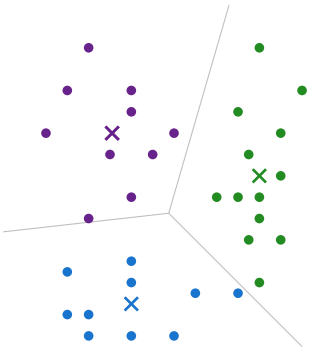


Figure 1.3: Possible outcome of k-means. Three clusters with data points ($\bullet$) and centers ($\times$).

subset's center m_i afterwards. As this procedure converges to a (locally) optimal solution, Lloyd's algorithm terminates as soon as the assignment remains unchanged from one step to another, see Figure 1.3. Being a heuristic, Lloyd's algorithm comes with disadvantages like the possibility for the search to get stuck in a local optimum and, thus, to miss approaching a global one. This issue as well as shortcomings of the clustering approach itself, like the need to define k manually, inspired several extensions and variants of the original k-means approach [Jai10]. Arthur and Vassilvitskii [AV07], for example, provide the extension k-means++. They improve the initialization of the original k-means algorithm and, thus, get a bound for the expected quality of the solution found which depends on $\ln k$ only.

Ester et al. [EKSX96] sought for fast algorithms in order to cluster large data sets. Existing approaches were either not satisfying with respect to their results, as many partitional algorithms yield convex clusters only, or were not fast enough. As a consequence, they came up with an algorithm for the density-based spatial clustering of applications with noise (in short: *DBSCAN*). Based on a distance function d and a corresponding threshold θ, both selected by choice, a neighborhood graph is set up. Its vertex set is formed by S. For every pair of objects that are θ or less apart (with respect to d), they introduce an edge. Every object that is connected with at least an also predefined minimum number m of other objects is considered to be a core object. Further, Ester et al. define border objects as those connected to at least one core object but, in total, only to a number of objects not exceeding m. Every other point is considered to be an outlier. Each connected component of core objects together with adjacent border objects forms a cluster. This way, Ester et al. presented a fast clustering algorithm that yields clusters of any suitable shape based on the density of the objects in the considered data set, see Figure 1.4. In a comparative study, for example, Cetinkaya et al. recognized DBSCAN as most effective for aggregating buildings in urban blocks. On the other hand, DBSCAN serves as a basis for various extensions such as SCAN, which improves the handling of vertices bridging different clusters, or many others [AAS10, KRA+14].

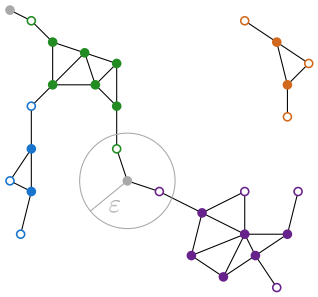


Figure 1.4: Possible result of DBSCAN for $m = 3$. Clusters can be identified by color (•: core points, ◦: border points), outliers are gray.

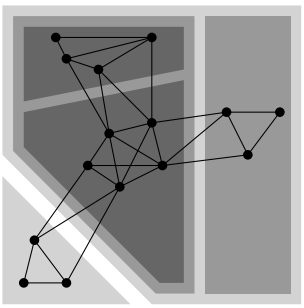


Figure 1.5: Recursive application of graph cuts (light to dark gray).

In contrast, Anders [And03] used multiple graphs for designing his graph-based approach for a hierarchical parameter-free graph clustering algorithm (*HPGCL*). The set S forms the vertex set of each considered graph. In the beginning, each vertex forms its own cluster. The edge sets of the graphs describe different levels of proximity. In such a graph, any two clusters connected via an edge between a pair of their vertices are candidates for merging. Anders considers the compatibility of these clusters with respect to density, distance, and their neighborhoods. The size of the considered neighborhoods is increased steadily by considering graphs with more and more edges. Since Anders uses five well-described and established graphs (Nearest Neighbor Graph $\subseteq$ Minimum Spanning Tree $\subseteq$ Relative Neighborhood Graph $\subseteq$ Gabriel Graph $\subseteq$ Delaunay Triangulation), his approach does not depend on any decision by the user besides defining the input set. Steiniger et al. [SBW06], for example, integrate a graph-based approach into their algorithm for detecting groups of islands automatically.

Shi and Malik [SM00] extend the concept of graph cuts, i.e., partitioning a graph into two sub graphs, to *normalized cuts*. Searching recursively for a minimum normalized cut splitting a subset of vertices, they find a partition of the graph's vertex set into multiple regions, see Figure 1.5. Shi and Malik designed their approach for image segmentation. It is an example of a spectral clustering algorithm based on the eigenvectors of a matrix derived from S. From a statistical point of view, Meilă and Shi [MS01] consider this approach as a Markov chain; Zhang et al. [ZHT06] use this approach to cluster trajectories in outdoor surveillance scenes.

Specific clustering problems and solutions Thomson and Brooks [TB02] consider road and river networks and present an algorithm for identifying chains of segments which follow a perceptual principle the authors call "good continuation", see Figure 1.6. For this purpose, they consider in particular every crossing and combine segments that participate in the respective crossing based on both geometric and non-geomtric criteria like the angle of

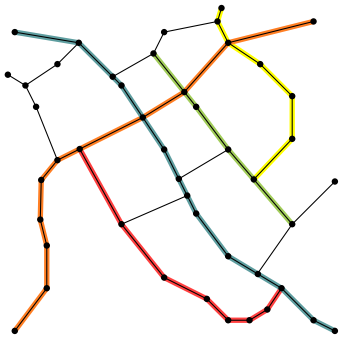


Figure 1.6: Road network with exemplary aggregation of road segments (black, between pairs of vertices) into strokes (in different colors).

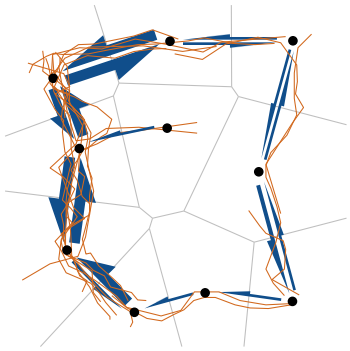


Figure 1.7: Exemplary clustering (blue) of trajectories (orange) based on a Voronoi tessellation of the plane (gray).

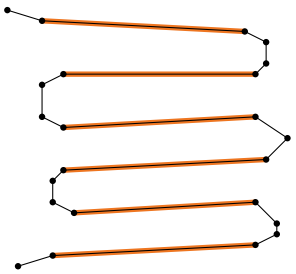


Figure 1.8: A single trajectory on a winding road. Orange segments may be erroneously identified as a cluster of trajectories.

deflection, type of road (for road networks), direction of flow (for river networks), or simply the feature's name. Yang et al. [YLL11] extend this concept by introducing hierarchies, which allows them to create strokes that are not connected. Thus, they are able to deal, for example, with dual carriageways and complex crossing.

Recently, Buchin et al. [BDGK19] contributed an algorithm for clustering trajectories. Their approach is based on the *k-center* problem, which is closely related to k-means. In contrast to k-means, a clustering is sought that minimizes the maximum occurring distance rather than the average distance. For this (**NP**-)hard problem [GJ90], exact and approximating algorithms exist [AP02]. Taking trajectories with an appropriate distance function, the Fréchet distance, as an input, Buchin et al. apply an existing approximation algorithm for point features [Gon85] after careful adaptations.

Andrienko and Andrienko [AA10] developed another algorithm that is capable of aggregating line features in order to present massive movement data. First, they extract interesting points along the considered trajectories including the start and end point as well as significant turns and stops. Subsequently, these points get clustered with a point clustering algorithm presented in the same work. It works similarly to k-means, but does not need a predefined number of clusters. Instead, a maximum (spatial) size is defined and considered. Based on the centers of the computed clusters, a Voronoi tessellation (see [dBCvKO08]) is computed, which segments the trajectories. Finally, for every adjacent pair of Voronoi cells, segments in between are aggregated respecting their direction, see Figure 1.7. They use the distance between original and aggregated trajectories as a measure of quality. Global minimization, however, does not take place.

Lee et al. [LHW07] likewise suggest a partition-and-group framework for aggregating trajectories. Inspired by DBSCAN, Lee et al. developed a density-based trajectory clustering algorithm (*TRACLUS*). After a line simplification, Lee et al. focus on the line segments forming the trajectories. Consequently, they need to beware of detecting multiple segments

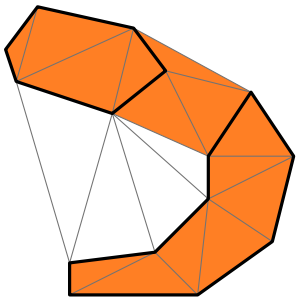


Figure 1.9: Orange polygon is the clustering result of applying adopt merge to two polygons (thick, black).

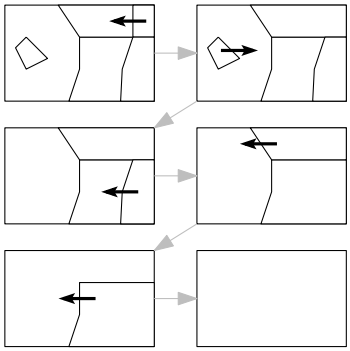


Figure 1.10: Region growing. The smallest region is merged with its largest neighbor. In general, size can be replaced by some measure of importance.

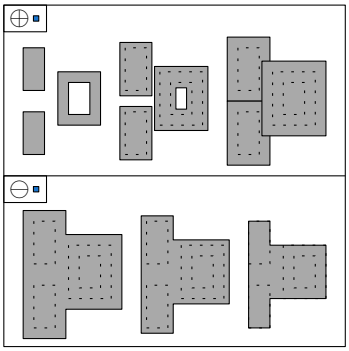


Figure 1.11: Morphological operators applied to a group of buildings. After two steps of dilation (top) with the small rectangle (blue), two steps of erosion (bottom) are applied.

of a single trajectory (or little more) as a cluster rather than noise, see Figure 1.8. For this purpose, they introduce a threshold defining the minimum number of different trajectories taking part in a detected cluster. Finally, Lee et al. compute a representative trajectory for each cluster found before. For both simplifying the given trajectories and setting the parameters for the DBSCAN-like cluster algorithm, Lee et al. suggest heuristic approaches. The quality of their approaches is assessed empirically; a precise problem formulation as an optimization problem, for example, is not given.

There are various problems concerning the aggregation of areal features. Ware et al. [WJB95], for example, presented four operators for aggregating possibly disjoint polygons with the help of a triangulation-based data structure. Three of these operators describe the process of stitching two polygons with different preprocessing steps, i.e., optional shifting or rotating of one of the given polygons. The operator *adopt merge* aggregates two polygons including free space in between. This results in a larger polygon containing both original ones; see Figure 1.9. Ware et al. focus on presenting the operators rather than assessing the quality of their results.

In contrast to Ware et al., van Oosterom [vO95] does not consider separate polygons but a partition of the plane into polygons. Van Oosterom tackles a selection problem in the context of map generalization and deals with gaps coming into being when not selecting individual polygons of the considered partition. He suggests filling each such gap with the most important adjacent region. Another way to articulate this procedure is to aggregate those two regions. Haunert and Wolff [HW10a] formalize this approach, terming it *region growing*, see Figure 1.10. They use this greedy approach as a first processing step of a heuristic approach for aggregating spatial units of the plane. This problem is at the core of a group of problems considered in a variety of fields like school or political districting [CSGW04, GN70], land-use allocation [LZCJ08], or forest management [CCG+13]. These problems vary with respect to their focus on, for example, compactness of resulting regions, their contiguity,

or similarity with respect to a certain non-geometric, context-dependent attribute. Often, these problems are considered as optimization problems [GN70, Shi09] and tackled with both heuristic and exact algorithms [HW10a]. In Chapter 4, we present an improved exact solution to this problem.

Damen et al. [DvKS08] identify some of the aforementioned approaches as important for the aggregation of buildings based on building footprints. They contribute an approach that applies morphological operators, a concept closely related to Minkowski sums, to both simplify and aggregate groups of buildings, see Figure 1.11. Damen et al. asses the results of their approach visually.

Finally, aggregation algorithms have been designed to aggregate objects of higher dimensions than areas. Like Damen et al., Kada [Kad10] uses morphological operators to design an aggregation algorithm for 3D buildings. He adapts the concept to 3D and applies the operators iteratively in order to effectively transfer an earlier developed simplification algorithm for buildings to groups of buildings. Guercke et al. [GGBS11] design aggregation algorithms for buildings based on another 2D aggregation algorithm. They adapt and extend approaches developed for the aggregation of areal units in a partition of the plane [HW10a].

Aggregation of non-geometric data Aggregation problems do not necessarily involve geometric information. In particular, in the process of decision-making, aggregation is useful whenever multiple criteria influence a decision. Yager [Yag88] introduces ordered weighted averaging (OWA) operators that provide a multitude of aggregation operators beyond demanding that all criteria or that at least one criterion must be fulfilled. Aggregation of criteria is one option to deal with multiple criteria; an example for its application using spatial data is multi-criteria route planning [ND11].

Conclusion Several of the approaches above have in common that they provide rather universally applicable tools for the aggregation of spatial data. Hence, it is clear that the solutions they provide cannot be of optimal quality for every case of application. Most of these approaches, however, are designed to overcome shortcomings of existing approaches. As a consequence, for certain applications, some algorithms yield better results than others.

We sketched various approaches tailored to specific problems. Some of these approaches have been evaluated by experts sifting through produced results. This evaluation is surely convincing for the considered examples, but gives only little insight into the quality of the presented algorithm in general. In this thesis, we present spatial aggregation problems as optimization problems. That means, for every considered problem, we introduce a mathematical function evaluating a solution's quality. Subsequently, by designing problem specific solutions rather than applying good and established generic tools, we develop algorithms that solve these optimization problems optimally.

1.3 Optimization

As indicated in Section 1.1, the evaluation of the results of the generalization process depends, inter alia, on subjective criteria. Li and Openshaw [LO93] consider this a major obstacle on the way to an automated generalization process. The generalization operator of aggregation is no exception. Subjectivity in aggregation is a phenomenon which gets particular attention when analyses are run on a partition of the plane, i.e., when the area of investigation is subdivided into smaller areas. For this case, Openshaw [Ope84] coined the term *modifiable areal unit problem* (MAUP) and summarized this problem as follows:

> "(...) the areal units (zonal objects) used in many geographical studies are arbitrary, modifiable, and subject to the whims and fancies of whoever is doing, or did, the aggregating."

A basic example for this problem is depicted in Figure 1.12. Here, 25 units with a binary attribute (in this example: *gray* or *white*) are given. Now, areas are sought that generalize this data. These areas are limited to the same attribute values. Even demanding a fair distribution of five units per resulting area leads to extremely varying outcomes. Figures (a) and (b) depict aggregations yielding the same distribution of the attributes for the resulting regions as the input data. Figure (c) put an overwhelming majority of the attribute *gray* into the reader's mind. In Figure (d), the majority ratio gets inverted.

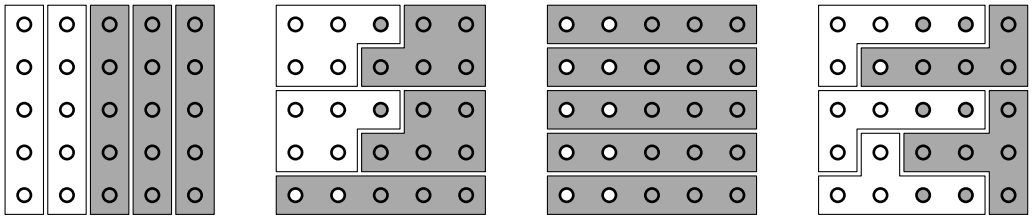


Figure 1.12: Example for MAUP. 25 input units with a binary attribute (*gray*/*white*) are aggregated into 5 areas of 5 units each. Assigning the attribute of the majority of units to the containing area results in different distributions. (a)/(b) Like in the input, 2/5 of areas are *white*, 3/5 are *gray*. (c) Designing the areas such that each contains one more *gray* unit than *white* units results in 100% areas with the attribute *gray*. (d) Designing the areas such that as many as possible contain one more *white* unit than *gray* units results in an overall majority of areas with the attribute *white* not representing the original data.

This problem captures the attention of the general public when elections are on the agenda. This is the case in particular if each district receives one representative depending on the relative majority within. In this context, the problem is known as *Gerrymandering* and as such a popular research topic in geoinformation science; Ricca et al. [RSS13] reviewed existing approaches. Although not all research work done focuses on non-partisan political districting [Nag65, She98], this application example underlines the necessity of means for objective aggregation [MJN98, RS08].

One way to achieve objective aggregation is via optimization. Papadimitriou and Steiglitz [PS82] suggest the following definition.

Definition 1.1. *An* instance of an optimization problem *is a pair* (F, c)*, where* F *is any set, the domain of feasible points;* c *is the* cost function*, a mapping*

$$c\colon F \to \mathbb{R}.$$

The problem is to find an $f \in F$ *for which*

$$c(f) \leq c(f') \qquad \text{for all } f' \in F.$$

Such a point f *is called a* (globally) optimal *solution to the given instance.*

Accordingly, F describes a set of valid solutions. Defining a cost function c, a set $O \subseteq F$ stands out that is optimal with respect to c. According to Definition 1.1, the elements of O cause minimal costs. Hence, in this case, the optimization problem is called a *minimization problem*. Likewise, we can define a *maximization problem*. Then, it is reasonable to call c a *score function* rather than the cost function. In general, we call c the *objective function* of the optimization problem.

Optimization is not immune to subjective criteria since a biased objective function will nevertheless create optimal solutions. However, subjectivity is harder to hide as the goal of an optimization approach needs to be defined concretely in the objective function.

Regardless of the motivation behind a model, optimization approaches allow assessing the model's quality. An evaluation of optimal solutions found reveals weak spots of the model applied. If an optimal solution does not correspond to the expected solution, refining the model, in particular the objective function, may be necessary. Thus, the evaluation of an optimal solution can lead to an improved model which, then again, leads to improved results (not with respect to the cost or score of the optimal solution).

For this evaluation, however, it is mandatory to find an optimal or at least a "good" solution. An algorithm that yields an optimal solution with certainty is called *exact*. Exact algorithms, however, may not always be applicable. In Section 2.1, we give a short introduction into the (time) complexity of problems and the running time of algorithms. In particular, we present a class of problems, **NP**-hard problems, for which the existence of efficient algorithms is unlikely. Hence, for some scenarios, finding an exact solution may take too much time. If this is not due to a hard-to-solve problem, this may be caused by a time-sensitive application. In any case, high solution quality needs to be traded off against low computation time since sufficiently efficient and exact algorithms may not be available.

Non-exact approaches can be summarized with the following two categories [PS82].

- An algorithm is called *heuristic* if it cannot give any guarantee with respect to the quality of the solution found.
- An algorithm is called an *approximation algorithm* if it guarantees solutions of a certain quality. Considering a minimization problem with an optimal solution m, for

example, an *ε-approximate algorithm* yields a solution $f \in F$ with a cost $c(f)$ such that

$$\left|\frac{c(f) - m}{m}\right| \leq \varepsilon$$

holds for any instance of the problem. That means, the relative error is bound by some $\varepsilon \in \mathbb{R}_{\geq 0}$.

Despite yielding results that are not necessarily optimal, both approximation algorithms [Chr76b, DMM+97] and heuristics [DAR12, Hau07, Ope77] are popular alternatives to slow exact algorithms. Zanakis and Evans [ZE81] give an extensive general overview for reasons why and how to use heuristic approaches.

The lower bound on the quality of the computed solution makes approximation algorithms attractive. However, even for small ε, the existence of an ε-approximation algorithm does not exclude the possibility that there is a heuristic approach that performs better on a certain problem. Also, in many cases, heuristic approaches are easier to design: For certain problems, the existence of efficient approximation algorithms is unlikely regardless of the targeted approximation, i.e. ε [PS82]. Furthermore, there are general heuristic approaches that are applicable to a multitude of problems. These so-called *metaheuristics* include the idea of a *local search*. This concept comprises different iterative strategies for searching a problem-specific defined neighborhood of a current solution for improvements. Depending on the searching strategy and the definition of the neighborhood, this results in a more or less sophisticated trial-and-error approach that is surprisingly successful in practice [PS82].

Nevertheless, in particular for heuristic approaches, it is difficult to predict or even evaluate the quality of the approach in practice. If no optimal solution is known, it is hard to assess a solution yielded by a heuristic algorithm. Consequently, it is reasonable to work on the development of exact optimization algorithms or the improvement of their running time. This is also the case if exact approaches are too slow for proper application. No matter how time sensitive the use-case is, during the development of fast, non-exact algorithm there is, in general, enough time to run slow algorithms that produce solutions of higher quality as benchmarks. This is a common strategy to evaluate heuristic approaches [RU01]. Rardin and Uzsoy [RU01] note, however, that this evaluation scheme becomes less sound with increasing complexity of the problem. In particular for **NP**-hard problems, only rather small instances are solvable exactly. Rardin and Uzsoy doubt the scalability of heuristics and, thus, suggest to evaluate heuristics by running comparisons on examples that are comparable in size to real-world applications even if this means relinquishing optimal solutions as benchmarks.

1.4 Goal and outline of this thesis

In this thesis, we aim for the development and the analysis of exact approaches to aggregation problems as they occur in the field of geoinformation science.

Talking about exact approaches makes sense only if the problems considered are handled as optimization problems. Thus, every problem that is dealt with in this dissertation needs to be formalized as an optimization problem first. This formalization is done either as a mathematical programming formulation or, in short, summarized as an unambiguous question.

Subsequently, we develop algorithms that solve the presented problems. We aim for a description and analysis of these algorithms that measure up to a high standard with respect to mathematical precision. Furthermore, our analysis comprises the verification of the correctness of our approach as well as an examination of its running time. If possible, we undertake a theoretical analysis of the asymptotic behavior of the running time. This is the case only if insight into all sub procedures is provided. In some cases, we apply proprietary software for solving linear programs and, thus, limit ourselves to an empirical running time analysis.

As the publication of the algorithms forming the backbone of this thesis has been addressed to an audience of experts, partly in conference proceedings with limited space, we assumed a rather high level of prior knowledge. In Chapter 2, we give a short recapitulation of background information on computational complexity, graph theory and mathematical programming. Besides, this chapter serves the purpose of clarifying our perception of fundamental concepts applied in the following chapters.

In Chapters 3 to 6, we present some of the algorithms developed during the past years. They have been published in journals or presented at conferences and then published in the corresponding proceedings:

Chapter 3	T. C. van Dijk, J.-H. Haunert, and J. Oehrlein. Location-dependent generalization of road networks based on equivalent destinations. *Computer Graphics Forum*, 35(3):451–460, 2016. doi:10.1111/cgf.12921
Chapter 4	J. Oehrlein and J.-H. 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
Chapter 5	J. Oehrlein, B. Niedermann, and J.-H. Haunert. Analyzing the Supply and Detecting Spatial Patterns of Urban Green Spaces via Optimization. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 87(4):137–158, 2019. doi:10.1007/s41064-019-00081-0
Chapter 6	J. Oehrlein, A. Förster, D. Schunck, Y. Dehbi, R. Roscher, and J.-H. Haunert. Inferring routing preferences of bicyclists from sparse sets of trajectories. In *Proc. 3rd International Conference on Smart Data and Smart Cities*, volume IV-4/W7 of *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 107–114, 2018. doi:10.5194/isprs-annals-IV-4-W7-107-2018

Finally, in Chapter 7, we conclude this thesis by reflecting its goals and giving an outlook with an overview of problems that remained open.

2 Methodological background

In this chapter, methodological backgrounds are presented. In particular, we study fundamentals that are required but not explained in the publications which are in the focus of this thesis. In Section 2.1, the basics of computational complexity theory are imparted. The high complexity of some examined problems is the reason for questioning exact approaches, i.e., approaches that yield a guaranteed optimal solution in the first place. In Section 2.2, fundamental concepts of graph theory are introduced as the problems we deal with in this work are modeled with the help of graphs. Finally, in Section 2.3, an introduction to mathematical programming and integer linear programming in particular is given. Existing sophisticated tools for solving integer linear programs often make integer linear programming the first choice to handle problems of high computational complexity.

2.1 Computational complexity theory

The focus of this thesis is on exact optimization algorithms. This automatically raises the question of why one should forego exact solutions when dealing with optimization problems. A reason for contenting oneself with a non-exact solution for a certain problem may lie in the problem's computational (time) complexity. This section provides insight into the field of complexity theory as it can be found in more detail in classic textbooks on algorithms [CLR90].

Time complexity of problems The *(time) complexity* of a problem is defined by the algorithms that are capable of solving the problem. In particular, the algorithm that solves a problem the quickest is of great importance for the problem's complexity. Hence, analyzing the running time of an algorithm is of major interest.

The *running time of an algorithm* is the number of primitive operations that need to be undertaken to process an instance. It is expressed independently from its implementation; it merely depends on the size of the input which itself is expressed in a problem-specific way. In general, in particular for the problems discussed in this dissertation, the size of the input is the number of items in the input data. The running time of an algorithm for sorting a set of $n \in \mathbb{N}$ items, for example, is expressed as a function in n. Problems that are defined with the help of a graph G, a data structure described in more detail in Section 2.2.1, often have running times that depend on two values, the number n of vertices and m of edges in G expressing the size of G. In that case, the running time is given as a function in n and m.

Analyzing algorithms, one notices that there is no such thing as a single running time for an algorithm. Depending on the nature of the given instance, the running time of an algorithm varies. For some sorting algorithms, for example, sorted instances are handled faster than

chaotic or reverse sorted instances. Hence, it makes sense to differentiate between best-case and *worst-case running times*. The worst-case is of particular interest as it gives an upper bound for the running-time that is valid for any instance of a fixed input size.

Distinguishing these cases does not allow computer scientists to give exact running times for algorithms. The following example indicates why this is often neither possible nor desired. In general, sorting algorithms are based on comparing objects. Comparing integer values is easier and can be done with less primitive operations than comparing two-dimensional point objects. This means, although the algorithm describes a general procedure for sorting objects, its exact running time depends on the very objects that are meant to be sorted. Hence, it is common in computer science to express running times with bounds rather than exactly. This is often achieved by means of *big $\mathcal{O}$ notation*.

Definition 2.1 (Big $\mathcal{O}$ notation)**.** *The set*

$$\mathcal{O}\big(g(n)\big) = \big\{ f(n) \;\big|\; \exists c, n_0 \in \mathbb{N}_0 \colon 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \big\}$$

describes the set of functions with a growth rate not significantly higher than the one of $g(n)$.

Algorithms with a running time in $\mathcal{O}(\log n)$, for example, are said to run in logarithmic time. The function in n expressing the running time grows with a rate that is not significantly higher than the growth of $\log n$. Likewise, algorithms with a worst-case running time in $\mathcal{O}(n)$ or $\mathcal{O}(n^2)$ are called linear-time or, respectively, quadratic-time algorithms. Furthermore, the term *polynomial-time algorithm* describes algorithms with a running time in $\mathcal{O}\big(p(n)\big)$ where $p(n) = \sum_{i=0}^{k} a_i n^i$ is a polynomial in n. Only polynomial-time algorithms are considered to be *efficient*.

Within the context of big $\mathcal{O}$ notation, there are further tools to analyze algorithms with respect to their running times. However, for determining the complexity of a problem, an algorithm solving this problem with the lowest upper bound for the worst case running time is decisive.

Complexity classes of decision problems In complexity theory, two complexity classes of problems play an important role. Formally, however, these classes are defined for *decision problems* only, i.e., problems that have one of two possible solutions: either "yes" or "no".

On the one hand, there is the class **P** containing decision problems that are regarded as tractable. These are problems to which a solution can be found efficiently, i.e., within polynomial time.

On the other hand, there is the class **NP**. For **NP**, we refer to an informal definition by Garey and Johnson [GJ90][1]:

> The class **NP** is defined informally to be the class of all decision problems Π that, under reasonable encoding schemes, can be solved by polynomial time nondeterministic algorithms.

[1] A detailed introduction into the sophisticated and well-founded theory behind these complexity classes can be found in a variety of sources [CLR90, GJ90, NW88].

Here, a nondeterministic algorithm describes an algorithm that first guesses a solution and then verifies or falsifies the guess in a deterministic way. A nondeterministic algorithm solves a problem in polynomial time if checking the guess takes polynomial time in size of the input.

Although **P** and **NP** are defined for decision problems only, they play an important role for the analysis of optimization problems. Consider an optimization problem Π, i.e. a problem that demands minimizing (or maximizing) a value. In that case, Π can be cast to a decision problem by introducing a bound k: Is there a solution to Π with an objective value less (or greater) than k? Papadimitriou and Steiglitz call this the *recognition version* of an optimization problem [PS82]. If an algorithm exists that solves the optimization problem in polynomial time, then its recognition version is in **P**. The optimal solution *opt* is computed in polynomial time. Since the comparison of *opt* and k is done in constant time, i.e. $\mathcal{O}(1)$, a decision is made in polynomial time.

Since an algorithm that computes a solution to a decision problem in polynomial time can be used to verify the same, $\mathbf{P} \subseteq \mathbf{NP}$ holds. However, it is unknown, whether $\mathbf{P} \neq \mathbf{NP}$ or $\mathbf{P} = \mathbf{NP}$ holds. This question is one of the currently best-known mathematical problems [CJW06].

In this context, a certain subclass of **NP** becomes particularly interesting. The class **NP**-complete contains all problems in **NP** that are **NP**-hard, i.e., problems that are at least as hard as the hardest problems in **NP**. As a consequence, an algorithm that solves an **NP**-complete problem efficiently can be used to solve any **NP**-complete problem efficiently since they are of the same complexity. In 1971, the Cook-Levin theorem [Coo71] produced the first member of **NP**-complete. Ever since, numerous **NP**-complete problems have been introduced [Kar72, GJ90]. Thus, there are a lot of problems that caught the interest of renowned scientists and none of them was able to decide whether an efficient algorithm exists let alone to produce one.

In Chapters 3 and 4, we deal with **NP**-complete problems. In order to provide an exact algorithm to the problem described in Section 4, we apply means established in the field of combinatorial optimization, i.e. mathematical optimization on discrete sets [PS82]. In Section 3, we identify the problem under consideration as a special case of an **NP**-complete which happens to be solvable efficiently.

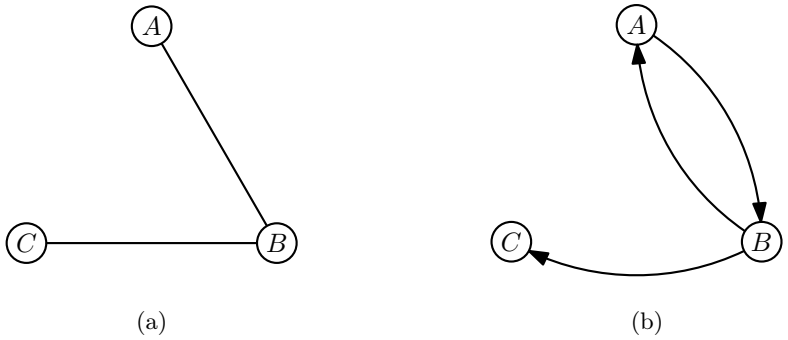
2.2 Graph theory

In this section, basic concepts and algorithms from graph theory that are fundamental for the following chapters are presented. A more detailed introduction to graph theory can be found in textbooks on algorithms [CLR90]. A graph is a data structure that is suitable to record, manage and analyze relations between objects. In particular, it is suitable for spatial relations and, hence, used throughout this dissertation.

In Section 2.2.1, fundamental definitions and basic concepts of graphs are presented. In particular, we introduce a notation that is used in the following chapters. Then, in Section 2.2.2, fundamental algorithms dealing with graphs are introduced. Finally, in Section 2.2.3, flow networks are introduced, a concept for modeling commodity flow based on graphs.

2.2.1 Basic concepts

Graphs and subgraphs A *graph* G is an ordered pair (V, E) of a *vertex* set V and an *edge* set E. In general, each element of E consists of two elements of V. In an *undirected graph* (see Fig. 2.1(a)), an edge between two vertices u and v is formed by a two-set $\{u, v\}$. In a *directed graph* (see Fig. 2.1(b)) the edge set is formed such that $E \subseteq V \times V$ holds. Thus, the directed edge from u to v is denoted as (u, v). In this dissertation, loops, i.e. edges from vertices to themselves, are not considered. Thus, two vertices u and v participate in an edge $\{u, v\}$ (or (u, v)); we call u and v *adjacent*. In undirected graphs, adjacency is symmetric and u and v are called *neighbors*; the set of all vertices adjacent to a vertex u, i.e. $\big\{v \in V \big| \ \{u, v\} \in E\big\}$, is called the *neighborhood* $N(u)$ of u. Conversely, we say an edge $\{u, v\}$ (or (u, v)) is *incident* to the vertices u and v. For a vertex u, the *degree* is defined as the number of incident edges. In a directed graph, the degree of a vertex u is the sum of the in-degree, the number of *incoming* edges $(\cdot, u)$, and the out-degree, the number of *outgoing* edges $(u, \cdot)$.



(a) (b)

Figure 2.1: Visualization of exemplary graphs with vertex set $V = \{A, B, C\}$. (a) Undirected graph with edge set $E = \big\{\, \{A, B\}, \{B, C\} \,\big\}$ (b) Directed graph with edge set $E = \big\{\, (A, B), (B, A), (B, C) \,\big\}$

Let $V', V^* \subseteq V$ be subsets of V. In the following, $E|_{V'}$ denotes the set of edges in E between vertices of V', i.e. $\big\{\{u, v\} \in E \big| \, u, v \in V'\big\}$ (or $\big\{(u, v) \in E \big| \, u, v \in V'\big\}$ in the directed case). Furthermore, for directed graphs, we write $E|_{V' \to V^*}$ for the set of edges in E from vertices in V' to vertices in V^*, i.e. $\big\{(u, v) \in E \big| \, u \in V', v \in V^*\big\}$.

A graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E|_{V'}$ is called a *subgraph* of $G = (V, E)$. If further $E' = E|_{V'}$ holds, G' is the subgraph of G *induced by* V'. If G' has a particular property and there is no V^* with $V' \subsetneq V^* \subseteq V$ inducing a subgraph with this property, we call G' a *maximal subgraph* (with respect to said property).

Paths and reachability A *path* P from a *source* s to a *target* t is a sequence of vertices $\langle v_0, \ldots, v_k \rangle$ with $v_0 = s$ and $v_k = t$ such that $\{v_i, v_{i+1}\} \in E$ for $0 \leq i < k$. If $s = t$ holds, the path P is called *cycle*. In particular for $s \neq t$, we refer to P also as an *s-t path* and denote P_{st} if its source s and target t are of importance. We call a path *simple* if its vertices are pairwise different. The path P is composed of the edges $\{v_0, v_1\}$, $\ldots$, $\{v_{k-1}, v_k\}$ and

contains the corresponding vertices. The *length of a path* is its number of edges, here k. A subpath of P is a path formed by vertices $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ with $0 \leq i \leq j \leq k$. The concept of paths works similarly in undirected and directed graphs.

A vertex v is *reachable* from a vertex u if a u-v path exits. In undirected graphs, reachability is reflexive, symmetric, and transitive, i.e., an equivalence relation. An undirected graph is *connected* if for every pair of vertices $u, v \in V$ the vertex v is reachable from u. In this context, we are also interested in the maximal connected subgraphs, the *connected components* of a graph. The connected components of an undirected graph are its equivalence classes with respect to reachability.

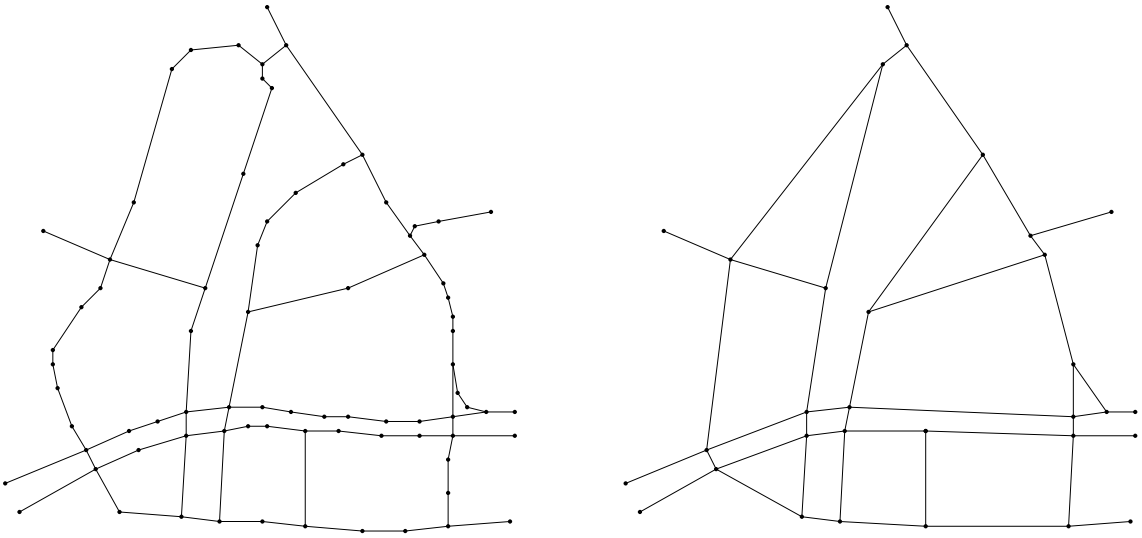
Further concepts in graph theory Often, graphs are enriched with a *weight function* yielding additional information on the data organized by the graph. A mapping $w\colon V \to \mathbb{R}$ is called a *vertex-weight function*. An *edge-weight function* $w'\colon E \to \mathbb{R}$ yields additional information on the relation between two vertices u and v in V that is described with an edge $\{u, v\} \in E$. It is common to refer to a graph combined with a corresponding edge-weight function as a *weighted* graph. We extend this definition to paths by defining the *weight of a path* as the sum of the weights of its edges. An s-t path of minimum weight often is called a *shortest path*. A path of minimum length is a shortest path assuming uniform weights of the edges of G.

A connected graph without cycles is called *tree*. A *rooted tree* is a tree in which one vertex r is designated as the *root*. As there are no cycles in trees, there is a unique path from any vertex v to the root. This path's length, i.e. the number of edges between v and the root, defines the *depth* of v in the tree. Each vertex u on the r-v path is an *ancestor* of v and v is a descendant of u. The ancestor sharing an edge with v is called the parent of v. Every descendant of v sharing an edge with it is a *child* of v. Other children of v's parent are v's *siblings*. The root r is the sole vertex without parent. A vertex without children is called *leaf*. Any other vertex is an *internal vertex*. Given a graph $G = (V, E)$, any tree with the same vertex set V is called a *spanning tree* of G.

An undirected graph $G = (V, E)$ is called *complete* if $\{u, v\} \in E$ holds for every two vertices $u, v \in V$. It is called *bipartite* if V can be partitioned into V_1 and V_2 such that any edge $\{u, v\} \in E$ is incident to a vertex $u \in V_1$ and a vertex $v \in V_2$. A *complete bipartite* graph is a bipartite graph with $\{u, v\} \in E$ for every $u \in V_1$ and $v \in V_2$.

Example 2.1. *A typical example for graphs in geoinformatics is the digitization of road networks. For navigation tasks, such a graph is the foundation for computing shortest paths (see Section 2.2.2). Computing shortest paths in a road network is applied for solving various problems in this thesis (see Chapters 3, 5, and 6). The level of abstraction varies according to the application, see Fig. 2.2.*

Depending on the tackled problem, road networks are digitized as undirected or directed graphs. Considering one-way streets, for example, promotes modeling the network as a directed graph. It is common to digitize road networks as weighted graphs with an edge weight reflecting the distance between the corresponding vertices. Depending on the tackled problem, it is reasonable to consider as edge weights, for example, the geodesic distance between the vertices, their difference in height, or the travel time.



(a) (b)

Figure 2.2: Visualization of an exemplary road graph. (a) Road graph with geometric information as it is used, for example, for mapping. (b) Road graph with mainly topological information as it is used for shortest-path algorithms.

2.2.2 Graph algorithms

In this section, some fundamental algorithms dealing with graphs are presented. These algorithms explore the graph. That means, starting from a vertex s, such an algorithm visits its neighborhood $N(s)$. Then, the algorithm continues visiting neighboring and so far unexplored vertices. This way, the structure of the graph becomes visible and it is possible to express the relation between arbitrary vertices within the graph.

Breadth-first search *Breadth-first search* (BFS) describes such a fundamental algorithm. Given a graph $G = (V, E)$ and a vertex $s \in V$, the algorithm systematically explores the graph, beginning from s. During this exploration, the algorithm *visits* the vertices of a graph. Visiting a vertex, its adjacent vertices are *discovered* if they have not been discovered before. The order in which the vertices of a graph are visited is the order of discovery. Hence, after s, all vertices in $N(s)$ are visited; vertices for which the minimum length of a path from s, or the *distance*, is $d = 1$. Afterwards, their neighborhoods are explored, which results in discovering vertices with distance $d = 2$ from s. This continues until all vertices that are reachable from s have been visited, see Figure 2.3.

Cormen et al. [CLR90] describe BFS as it is presented in Algorithm 1. They use a *queue* Q to organize the order in which the vertices are visited. In a queue, elements can be added (*enqueue*) and extracted (*dequeue*). The order in which the elements are added is the same as the order in which the elements are extracted. Hence, a queue follows the first-in-first-out principle (FIFO). For every vertex u, they store and update the following values:

- The distance d from s, i.e. the minimum length of a path from s to u. The variable $d[u]$ is initially set to ∞.

- The predecessor π, i.e. the vertex $\pi[u]$ from which u is discovered during BFS.
- The color, i.e. a key giving information about the state of the vertex u. In its original state, *white*, a vertex has been neither visited nor discovered. After its discovery, a vertex' color is set to *gray*. Finally, after having been visited, the color of the vertex is set to *black*.

Algorithm 1: BFS(G, s)

Data: Graph $G = (V, E)$, vertex $s \in V$
Result: Every vertex G that is reachable from s is visited, distances d from s are computed and information on the minimum-length path from s is gained

```
1 // initialize, see Algorithm 2
2 color, d, π ←BFS_init(G);
3 // start exploration, see Algorithm 3
4 BFS_explore(s, G, d, π, color)
```

Algorithm 2: BFS_init(G, s)

Data: Graph $G = (V, E)$
Result: Distances, predecessors, and colors set to initial values, i.e., ∞, NIL, and *white*.

```
1 foreach u ∈ V do
2 |   color[u] ←white;
3 |   d[u] ← ∞;
4 |   π[u] ←NIL;
5 return color, d, π;
```

In order to determine the complexity of BFS, we first notice that the initialization step, see Algorithm 2, considers every vertex once and, thus, takes $\mathcal{O}(n)$ time, where $n = |V|$. Considering the exploration of the graph, i.e. Algorithm 3, we first note that every vertex is enqueued exactly once since it is enqueued only if it is *white* but it is colored *gray* directly afterwards, see Line 11. Hence, the while-loop (Line 6) considers every vertex reachable from s exactly once. Thus, there are n dequeue- and n enqueue-operations, each of which can be done in constant time. Consequently, every vertex's neighborhood is explored exactly once (for-loop in Line 8). Hence, each edge (u, v) is considered twice; on the one hand due to $(u, v) \in N(u)$, on the other hand due to $(u, v) \in N(v)$. Since all iterations of the while-loop consider each vertex at most once and all iterations of the for-loop consider each edge at most twice, the exploration takes $\mathcal{O}(n + m)$ time with $m = |E|$. The resulting overall running time is in $\mathcal{O}(n + m)$.

With the help of the predecessors π, it is possible to set up the *breadth-first tree* corresponding to G and s. This tree rooted at s contains all vertices of G and displays the paths of minimum length d, see gray tree in Figure 2.3. This tree is gained by adding every vertex $v \in V \setminus \{s\}$ to $\pi[v]$ as a child.

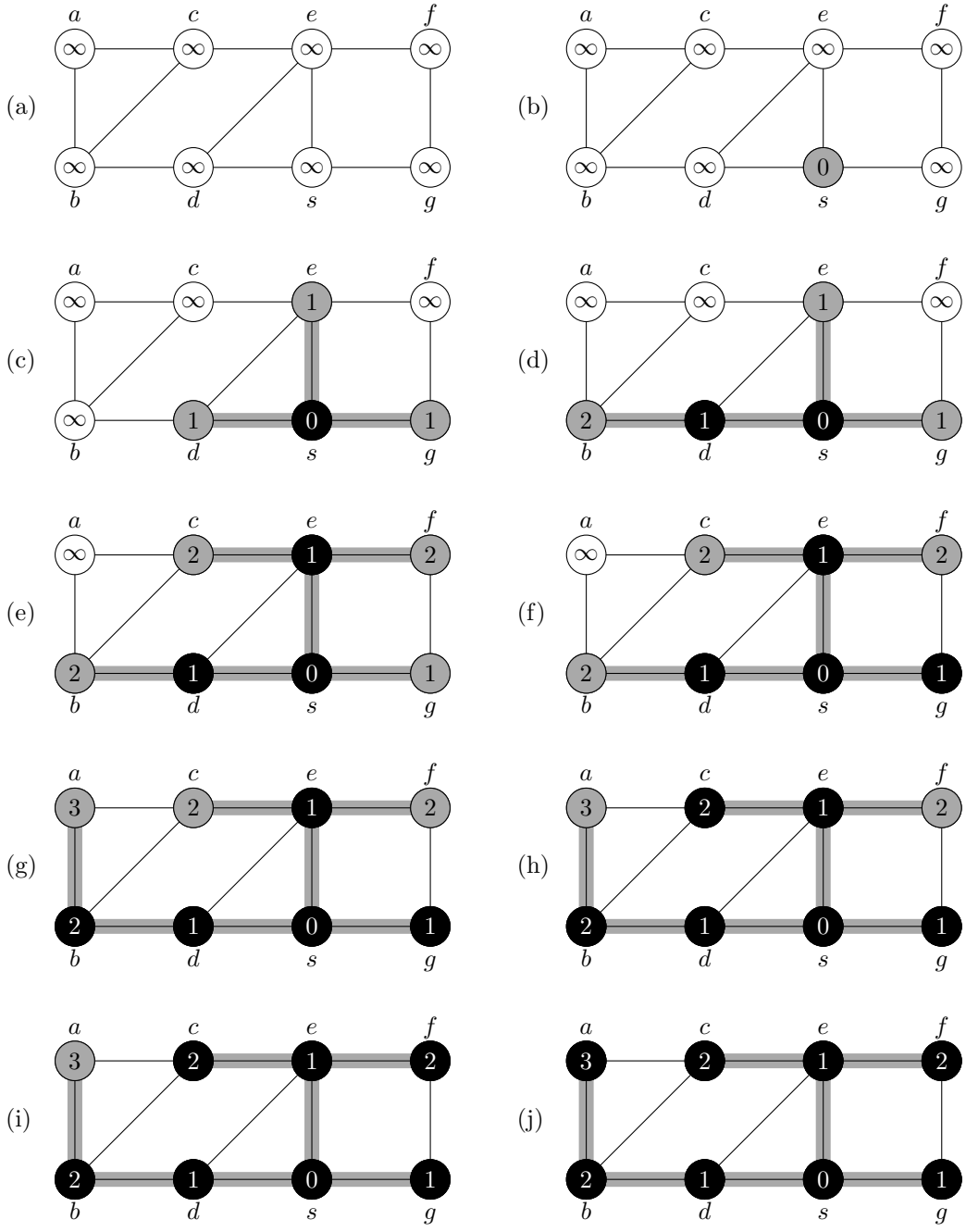


Figure 2.3: Breadth-first search on a graph starting from vertex s. The resulting breadth-first tree (gray lines) depends on the order in which the vertices are visited and, thus, how they are ordered in a neighborhood. Here, they are ordered alphabetically.

Algorithm 3: BFS_explore(s, G, d, π, color)

```
1  Q ← new Queue();
2  color[s] ←gray;
3  d[s] ←0;
4  π[s] ←NIL;
5  Q.enqueue(s);
6  while Q ≠ ∅ do
7      u ← Q.dequeue();
8      foreach v ∈ N(u) do
9          if color[v]=white then
10             Q.enqueue(v);
11             color[v] ←gray ;
12             d[v] ← d[u] + 1;
13             π[v] ← u;
14     color[u] ←black;
```

Depth-First Search Another fundamental concept for graph exploration is *depth-first search* (DFS). In contrast to BFS, DFS hardly gives information on the minimum length of a path between the start vertex s of the search and another vertex in the graph. The reason for this is that DFS drives an exploration path deep into the graph rather than exploring the surroundings of s first. DFS visits a so far unexplored neighbor of the most recently visited vertex. This procedure begins at the start vertex s and continues until it runs into a dead end, i.e. the DFS reaches a vertex the neighborhood of which consists only of vertices that have already been visited. In this case, the search continues from the most recently visited vertex with unexplored neighbors, see Figure 2.4.

Algorithm 4: DFS(G, s)

Data: Graph $G = (V, E)$, vertex $s \in V$
Result: Every vertex G that is reachable from s is visited

```
1 // initialize, see Algorithm 5
2 color, π ←DFS_init(G);
3 // start exploration, see Algorithm 6
4 DFS_explore(s, G, π, color)
```

Concerning the complexity of DFS, we follow the reasoning for the complexity of BFS. The initialization of DFS has the same complexity as the one of BFS. Since each vertex is colored *gray* as soon as it gets visited and only *white* vertices get visited, every reachable vertex is visited only once. Then, its complete neighborhood is considered. Hence, every edge (u, v) is considered at most twice; once from u and once from v. Consequently, the overall running time is in $\mathcal{O}(m + n)$ as well.

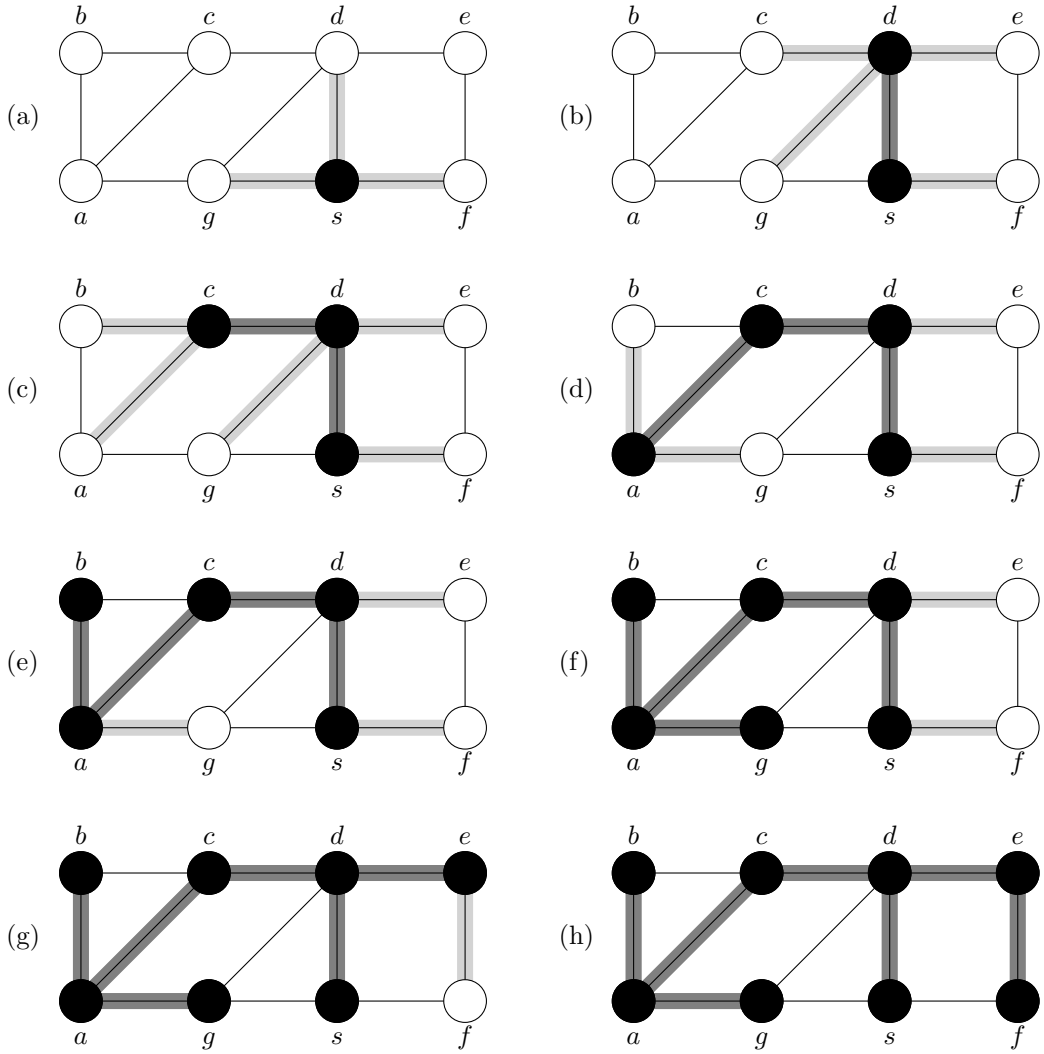


Figure 2.4: Exemplary run of a depth-first search. Starting from s, the exploration happens on a path that forces its way deep into the graph (dark gray). The next vertex to be visited is a vertex in the neighborhood of the foremost vertex on the path that has neighbors that have not been visited yet. It is selected depending on the order of vertices in the neighborhood. Here, vertices are ordered alphabetically. In Figure (e), the foremost vertex on the path b has no adjacent vertices that have not been visited so far. Hence, the search backtracks along the path until a vertex is found that fulfills this requirement. The vertex a has the neighbor g, which has not been visited yet. Afterwards, further backtracking is necessary and the search continues from d, see Figure (g).

Algorithm 5: DFS_init(G, s)

Data: Graph $G = (V, E)$
Result: Predecessors and colors set to initial values, i.e., NIL and *white*.

```
1 foreach u ∈ V do
2     color[u] ←white;
3     π[u] ←NIL;
4 return color, π;
```

Algorithm 6: DFS_explore(u, G, π, color)

```
1 color[u] ←gray;
2 foreach v ∈ N(u) do
3     if color[v]=white then
4         π[v] ← u;
5         DFS_explore(v, G, π, color);
6 color[u] ←black;
```

Finding connected components These search algorithms serve as a basis for further algorithms. A simple extension of the presented algorithms is suitable for discovering and counting connected components of a graph since both BFS and DFS visit only vertices that are reachable from the start vertex s. Algorithm 7 uses DFS for this task. After the exploration of the connected component of s, the search is started from the next vertex that has not been visited yet. Continuing this procedure yields a series of DFS calls that all started in different connected components. Thus, storing the start vertex of each DFS in a list grants access to every connected component of a graph. This list's length, on the other hand, gives information on the number of connected components in a graph. In particular, if the list contains only one element, i.e. every vertex is visited during the first DFS, the graph is connected.

Algorithm 7: connected_components(G)

Data: Graph $G = (V, E)$
Result: a list of representatives of the connected components of G

```
1 // initialize, see Algorithm 5
2 color, π ←DFS_init(G);
3 cc←list();
4 // start exploration
5 foreach s ∈ V do
6     if color[s]=white then
7         cc.add(s);
8         DFS_explore(s, G, π, color)          // see Algorithm 6
9 return cc;
```

Dijkstra's Algorithm Applying the concept of BFS to weighted graphs leads to shortest-path algorithms, i.e., algorithms producing paths of minimum weight. In particular, Dijkstra's Algorithm, the most common algorithm for computing shortest paths in graphs with non-negative weights, works analogously to BFS, see Algorithm 8. Like BFS, Dijkstra's Algorithm computes each vertex' distance d from s, which in a weighted graph is commonly interpreted as the minimum weight of a path from s. An important difference in the procedure is the order in which the vertices of the graph are visited. BFS visits the graph's vertices in non-descending order with respect to the minimum length of a path to the corresponding vertex. Analogously, Dijkstra's Algorithm takes the weight of the shortest path into account. The main issue is that the minimum-weight path does not necessarily coincide with the path of minimum length. Consequently, an early-discovered vertex may be visited late if it is connected to the rest of the graph via heavy-weight edges only.

For maintaining the order in which vertices are visited, a priority queue is used. In this abstract data structure, each object is associated with a key, for example the distance d. A priority queue provides the user in particular with a method *decreaseKey*, which updates the key of an object with a lower value, and a method *extractMin*, which returns the object with the currently minimal key.

Algorithm 8: Dijkstra(G, s)

Data: Graph $G = (V, E)$, vertex $s \in V$
Result: Every vertex G that is reachable from s is visited, distances d from s are computed and information on the minimum-length path from s is gained

```
1  // initialize, see Algorithm 2
2  color, d, π ←BFS_init(G);
3  // start exploration, compare Algorithm 3
4  Q ←new PriorityQueue(V);
5  color[s] ←gray;
6  d[s] ←0;
7  π[s] ←NIL;
8  Q.decreaseKey(s,0);
9  while Q ≠ ∅ do
10     u ← Q.extractMin();
11     foreach v ∈ N(u) do
12         if d[v] > d[u] + w(u, v) then
13             color[v] ←gray;
14             d[v] ← d[u] + w(u, v);
15             π[v] ← u;
16             Q.decreaseKey(v, d[v]);
17     color[u] ←black;
```

The colors play a similar role as in BFS. The initial state is *white*: the vertex has been neither discovered nor visited. When a vertex v is discovered from a vertex u, it is colored *gray* and its current distance is set to the distance of u plus the weight of the edge $\{u, v\}$.

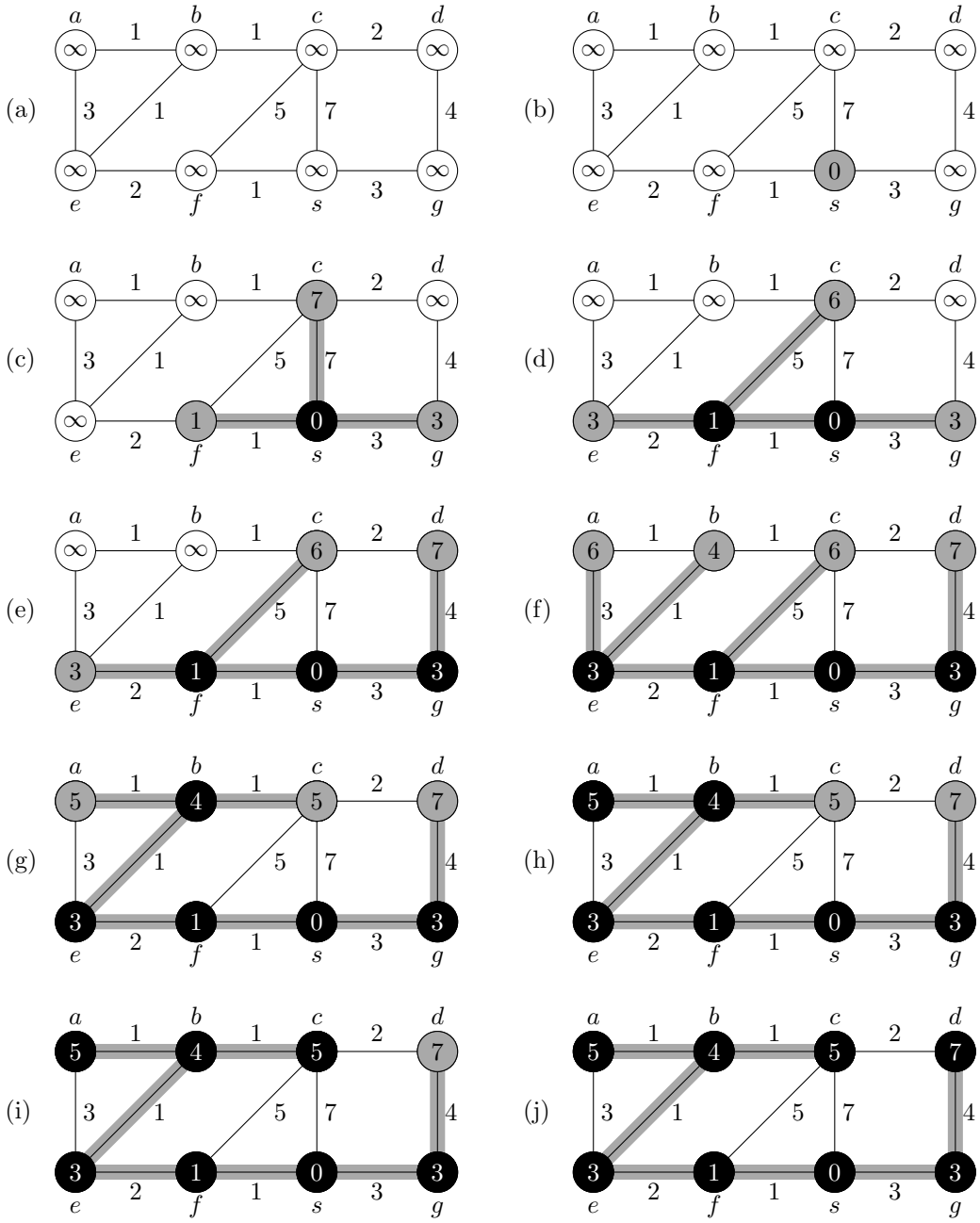


Figure 2.5: Applying Dijkstra's Algorithm on a weighted graph starting from vertex s. The resulting shortest-path tree is marked with thick gray lines.

Every time v is in the neighborhood of a visited vertex w, its current distance from s gets tested and updated if the path via w is shorter. Also, the key value of the vertex in the priority queue gets updated accordingly. This continues until v itself is visited and colored *black*. Then, the shortest path from s to v has been found, see Figure 2.5.

The running time of Dijkstra's Algorithm depends on the implementation of the priority queue. However, every vertex is extracted from the queue exactly once (line 10 in Algorithm 8). Furthermore, like in BFS, every edge is considered twice; *decreaseKey* in line 16 is called at most once per edge. Hence, in general, the algorithm takes $\mathcal{O}(n \cdot T_{dK} + m \cdot T_{eM})$ time with T_{dK} expressing the running time of *decreaseKey* and T_{eM} the running time of *extractMin*. Using an unordered list as a priority queue, for example, the running time is in $\mathcal{O}(m \cdot 1 + n \cdot n) = \mathcal{O}(n^2)$. With a proper data structure, namely a *Fibonacci-Heap*, the minimum can be extracted in amortized logarithmic time while decreasing keys is done in $\mathcal{O}(1)$ amortized time. Thus, a running time in $\mathcal{O}(n \log n + m)$ is possible.

With the information stored in π, i.e. information about the predecessor of a vertex, it is possible to build the shortest path. The equivalent of the breadth-first tree is the *shortest-path tree* which is the basis for our location-dependent generalization of road networks in Chapter 3.

2.2.3 Flow networks

Besides modeling the relation between objects, graphs are suitable for modeling network flows. The following introduction is based on the textbook by Ottmann and Widmayer [OW02]. Here, the flow of commodity through a network is simulated. This can be, for example, fluids running through pipes with sources and sinks, current flowing through electrical grids, or goods being transported from warehouses to stores. In these models, there are one or multiple *sources* feeding commodity into the network and *sinks* where the commodity leaves the network. On its way from source to sink, the commodity flow is limited by the *capacity* of the elements of the network. This can be, for example, the diameter of the pipe in a fluid network or the cargo capacity of trucks. Furthermore, commodity can only appear in the network through sources and can only disappear from the network through sinks.

Formal definition A *flow network* is modeled as a directed graph $G = (V, E)$ with a designated source $s \in V$, a designated sink $t \in V \setminus \{s\}$ and an edge capacity $c\colon E \to \mathbb{R}_{\geq 0}$. Since edges with no capacities do not contribute to the network, we consider $c(u, v) = 0$ equivalent to $(u, v) \notin E$. According to the model description above, we assume that for every $v \in V \setminus \{s, t\}$ there is a path P_{svt}. Otherwise, v does not play a role in the solution to the problem. We define a flow as function $f\colon E \to R_{\geq 0}$ with the following properties:

- *Capacity constraint*: For every pair $u, v \in V$, we require $f(u, v) \leq c(u, v)$.
- *Flow conservation*: For every vertex $v \in V \setminus \{s, t\}$, we require

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$$

While the capacity constraint guarantees that each edge's capacity is respected, the flow conservation makes sure that the inflow equals the outflow in every vertex that is neither source nor sink. Hence, no commodity can appear or disappear inside the network.

Figure 2.6 depicts a flow network with an exemplary flow f transporting commodity from s to t. The value $|f|$ of the flow gives the number of units that are transported, i.e. the number of units of commodity leaving s:

$$|f| \coloneqq \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s) \tag{2.1}$$

In the given example, there are $|f| = 2$ units of commodity transported.

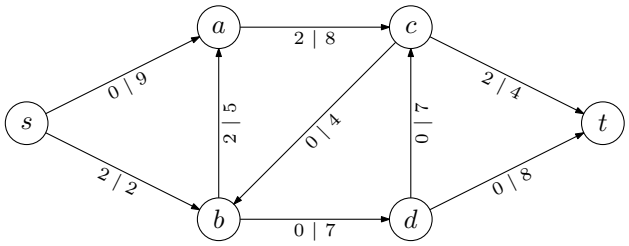


Figure 2.6: Example of a network flow. For each edge e, flow and capacity is given as $f(e) \mid c(e)$. There are 2 units of commodity transported that pass through the network along the path $\langle s, b, a, c, t\rangle$.

Maximum flow and cuts A common objective for network flows is to maximize the flow of the network. In order to estimate an upper bound for the *maximum flow*, we introduce the concept of cuts separating s and t. In a flow network, a *cut* (S, T) is a partition of the vertex set V into two sets S and T with $s \in S$ and $t \in T$. The edges from vertices of S to vertices of T, i.e. elements of $E|_{S\to T}$, are of particular interest. The *capacity* $c(S, T)$ of a cut is defined as the sum of the capacities of edges in $E|_{S\to T}$, i.e.:

$$c(S,T) \coloneqq \sum_{(u,v)\in E|_{S\to T}} c(u,v) \tag{2.2}$$

	MaximumFlow
Instance:	A directed graph $G = (V, E)$ with designated vertices $s, t \in V$. Capacities on the edges $c\colon E \to \mathbb{R}_{\geq 0}$. $x \in \mathbb{R}_{\geq 0}$.
Question:	Does there exist a flow f from s to t such that $\lvert f\rvert \geq x$ holds?

	MinimumCut
Instance:	A directed graph $G = (V, E)$ with designated vertices $s, t \in V$. Capacities on the edges $c\colon E \to \mathbb{R}_{\geq 0}$. $x \in \mathbb{R}_{\geq 0}$.
Question:	Does there exist a cut (S, T) with $s \in S$ and $t \in T$ such that $c(S, T) \leq x$ holds?

Likewise, we define the *net flow* $f(S, T)$ across a cut as

$$f(S,T) \coloneqq \sum_{(u,v)\in E|_{S\to T}} f\left(u,v\right) - \sum_{(u,v)\in E|_{T\to S}} f\left(u,v\right). \tag{2.3}$$

In the example in Figure 2.6, for $S \coloneqq \{s, a, b, d\}$ and $T \coloneqq \{t, c\}$, the capacity of the cut is $c(S, T) = 23$ while the flow across it is $f(S, T) = 2$. For $S' = \{s, a\}$ and $T' = \{t, b, c, d\}$, they are $c(S', T') = 10$ and $f(S', T') = 2$, respectively and we notice $f(S, T) = f(S', T') = |f|$. This holds in general.

Lemma 2.1. *Let f be a flow in a network $G = (V, E)$ with source $s \in V$ and sink $t \in V$. Let (S, T) be an arbitrary cut. Then, $f(S, T) = |f|$ holds.*

Proof. Let $1 \leq n < |V|$. Assume the claim is true for every cut $(S, V \setminus S)$ with $|S| = n$.

Now, consider $S' = S \cup \{w\}$ and $T' = T \setminus \{w\}$ with $w \in V \setminus S$. Then:

$$
\begin{aligned}
f(S',T') &= \sum_{(u,v)\in E|_{S'\to T'}} f(u,v) - \sum_{(u,v)\in E|_{T'\to S'}} f(u,v) \\
&= \sum_{(u,v)\in E|_{S\to T'}} f(u,v) + \sum_{(u,v)\in E|_{\{w\}\to T'}} f(u,v) \\
&\quad - \left(\sum_{(u,v)\in E|_{T'\to S}} f(u,v) + \sum_{(u,v)\in E|_{T'\to\{w\}}} f(u,v) \right) \\
&= \left(\sum_{(u,v)\in E|_{S\to T}} f(u,v) - \sum_{(u,v)\in E|_{S\to\{w\}}} f(u,v) \right) \\
&\quad + \left(\sum_{(u,v)\in E|_{\{w\}\to T}} f(u,v) - \sum_{(u,v)\in E|_{\{w\}\to\{w\}}} f(u,v) \right) \\
&\quad - \left(\sum_{(u,v)\in E|_{T\to S}} f(u,v) - \sum_{(u,v)\in E|_{\{w\}\to S}} f(u,v) \right) \\
&\quad - \left(\sum_{(u,v)\in E|_{T\to\{w\}}} f(u,v) - \sum_{(u,v)\in E|_{\{w\}\to\{w\}}} f(u,v) \right) \\
&= \sum_{(u,v)\in E|_{S\to T}} f(u,v) - \sum_{(u,v)\in E|_{T\to S}} f(u,v) \\
&\quad + \sum_{(u,v)\in E|_{\{w\}\to S\cup T}} f(u,v) - \sum_{(u,v)\in E|_{S\cup T\to\{w\}}} f(u,v) \\
&= f(S,T) \qquad \text{due to flow conservation and } V = S \cup T.
\end{aligned}
$$

By definition, $f(\{s\}, V \setminus \{s\}) = |f|$ holds, see Equation 2.1. Since $(S,T) = (\{s\}, V \setminus \{s\})$ is the only cut with $|S| = 1$, the claim holds for any cut. □

Lemma 2.2. *Let f be a flow in a network $G = (V,E)$ with source $s \in V$ and sink $t \in V$. Let (S,T) be an arbitrary cut. Then, $f(S,T) \leq c(S,T)$ holds.*

Proof. According to the definitions, the following holds:

$$
f(S,T) = \sum_{(u,v)\in E|_{S\to T}} f(u,v) \leq \sum_{(u,v)\in E|_{S\to T}} c(u,v) = c(S,T)
$$

□

Corollary 2.1. *Let (S,T) be an arbitrary cut. Then, $|f| \leq c(S,T)$ holds.*

Corollary 2.1 follows directly from Lemmata 2.1 and 2.2. Thus, the maximum flow is bounded by a cut of minimal capacity, a *minimum cut*.

Next, we introduce *residual networks* in order to show that the value of a maximum flow is actually the same as the capacity of a minimum cut. These networks also form the basis for standard maximum flow algorithms. For any flow f we can define a directed graph G_f. The vertex set matches the vertex set of the original flow network. The edge set is defined depending on the *residual capacity*. The residual capacity c_f describes the capacity that remains while considering the flow f. For any edge (u, v) in the network, it is defined as follows:

$$c_f(u, v) = c(u, v) + f(v, u) - f(u, v) \tag{2.4}$$

If $c_f(u, v) = 0$, we assume that the corresponding edge (u, v) is not contained in the residual network. Figure 2.7 displays the residual network corresponding to the flow network in Figure 2.6 with the flow defined there.

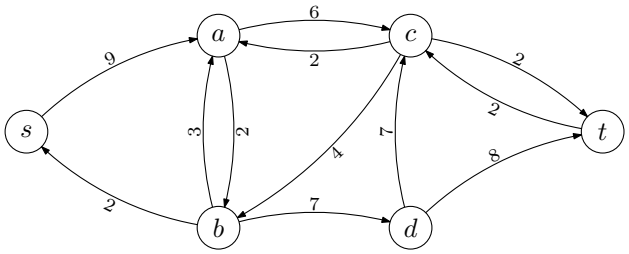


Figure 2.7: Exemplary residual network of the flow presented in Figure 2.6.

Since G_f is a graph, we can look for paths in it. In particular, we can look for s-t paths. For any s-t path $P_{st} = \langle s, v_1, \ldots, v_{n-1}, t\rangle$, we can increase f by $k = \min_{(u,v)\in P_{st}} c_f(u, v)$, the minimal residual capacity along the path. Updating G_f with respect to the increased flow, we can repeat this procedure with a different path since at least the edge defining k has no more capacity left. The following theorem is fundamental for the proof that this procedure, the Ford-Fulkerson method, see Algorithm 9, yields a maximum flow.

Theorem 2.1 (Max-flow min-cut theorem)**.** *Let f be a maximum flow in a network $G = (V, E)$ with source $s \in V$ and sink $t \in V$. Let $\mathcal{C}$ be the set of all cuts in G. Then*

$$|f| = \min_{(S,T)\in\mathcal{C}} c(S, T).$$

Proof. Since f is maximal, there is no augmenting path in the corresponding residual network G_f. Now, let S be the set of all vertices in G_f that are reachable from s and $T := V \setminus S$. Since there is no augmenting s-t path, $s \in S$ and $t \in T$ holds and (S, T) is a cut. Besides, for any $e \in E|_{S\to T}$ the equation $c_f(e) = 0$ holds. Due to the definition of c_f, see Equation 2.4, and the capacity constraint, $c(u, v) = f(u, v)$ holds. Consequently, $c(S, T) = f(S, T) = |f|$. Due to Corollary 2.1 the capacity of any cut is greater or equal to $|f|$. Hence, (S, T) is a minimum cut. □

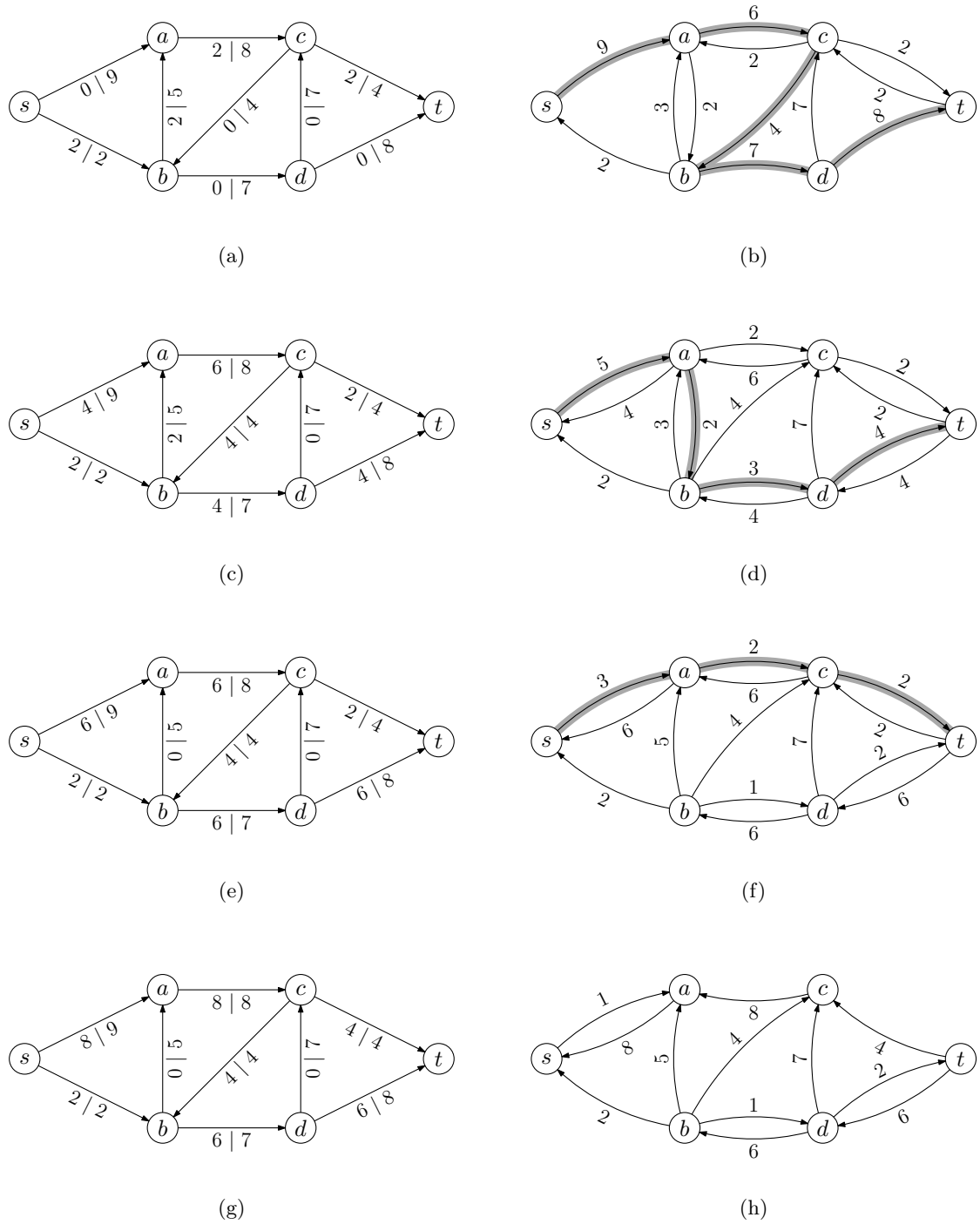


Figure 2.8: Flow network (left), residual network (right). Iterations from the initial flow (see Figure 2.6) to a maximum flow. Augmenting paths considered next are chosen arbitrarily and indicated as gray fat lines, see Figures (b), (d), and (f). Figures (c), (e), and (g) depict the current flow increased by a flow along the corresponding augmenting path. In Figure (h), no augmenting paths can be found as no edge from $S := \{s, a\}$ to $T := \{t, b, c, d\}$ is left. Hence, the cut (S, T) has a capacity corresponding to the current flow. According to Theorem 2.1, the cut (S, T) is minimal and the flow in Figure (g) is maximal.

Algorithm 9: Ford-Fulkerson-Method(G, c, s, t)

Data: Graph $G = (V, E)$ with capacity c, source $s \in V$, sink $t \in V$
Result: Maximum flow f from s to t

1 initialize flow f with $|f| = 0$;
2 **while** there is an augmenting path P **do**
3 augment f along P ;
4 **return** f;

There exist multiple algorithms following the procedure described in Algorithm 9. In Figure 2.8, a minimum cut and a maximum flow are computed for the network presented in Figure 2.6 with Algorithm 9; augmenting paths are selected arbitrarily. For flow networks with irrational capacities, the procedure does not even terminate if there is no adequate strategy for finding augmenting paths in line 2. Edmonds and Karp [EK72] use a BFS in order to find an augmenting path consisting of a minimum number of edges and guarantee that the algorithm terminates in $\mathcal{O}(n \cdot m^2)$ time. Modern strategies for determining augmenting paths improve the running time to $\mathcal{O}(n \cdot m)$ [Orl13].

In Algorithm 9, line 3, the flow is augmented along the path found in line 2 and the residual network is updated. When a maximum flow is found, no more augmenting paths exist. Then, the set X of vertices in V that are reachable from s in the residual network is part of a minimum cut $(X, V \setminus X)$. Since X can be found with a BFS in the residual network starting at s, the Ford-Fulkerson method thus also yields a minimum cut of a flow network.

So far, only flow networks with a single source and a single sink have been examined. However, Algorithm 9 is also applicable to *networks with multiple sources and/or multiple sinks*. For this purpose, the flow network is replenished with a *supersource* s' and a *supersink* t'. The supersource is linked to every actual source with edges of unlimited capacity. Likewise, the sinks are linked to the supersink, see Figure 2.9. Thus, the supersource feeds the actual source and the supersink drains the actual sinks. This way, a network with multiple sources and/or multiple sinks is emulated and the Ford-Fulkerson method can be applied to compute a maximum flow or a minimum cut.

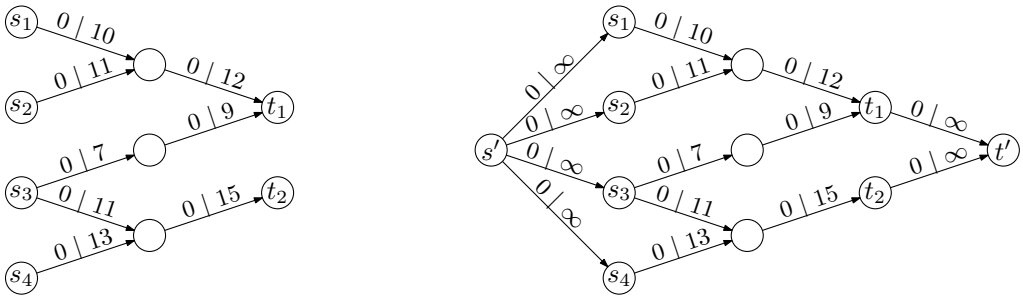


Figure 2.9: Simulating a maximum-flow problem with multiple sources and multiple sinks (Figure (a)) by means of a flow network with only one source and one sink (Figure (b)).

Application: Minimum-Weight Vertex Separators In the following, we describe the problem MinimumWeightVertexSeparator and an algorithm solving it. This is a subproblem of the problem that is further described in Chapter 4 and the following is excerpted from the work presented there [OH17].

Let $G = (V, E)$ be an undirected graph and $w\colon V \to \mathbb{R}_{\geq 0}$ a vertex-weight function. For $s, t \in V$, a *minimum-weight (s,t)-separator* $S \subseteq V \setminus \{s,t\}$ is a set of vertices with the following property: Every path in G from s to t contains at least one vertex in S. Furthermore, any set with this property has at least the same weight as S with respect to w.

	MinimumWeightVertexSeparator
Instance:	A graph $G = (V, E)$ with designated vertices $s, t \in V$. Weights on the vertices $w\colon V \to \mathbb{R}_{\geq 0}$. a number $w_0 \in \mathbb{R}_{\geq 0}$
Question:	Is there a vertex set $S \subseteq V \setminus \{s,t\}$ with a weight less than w_0 such that for any s-t path P_{st} the property $P_{st} \cap S \neq \emptyset$ holds?

If $s = t$ or $\{s,t\} \in E$ then there exists no such vertex and $S = \emptyset$. In any other case we make use of minimum (s,t)-cut algorithms and apply them to a modified graph $\tilde{G}$.

In order to use edge-cut algorithms we use G to build a directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$. Since only vertex weights are given and established edge-cut algorithms depend on edge weights, we need to define a mapping between vertices in G and edges in $\tilde{G}$. Thus, we introduce two vertices $\tilde{v}_0, \tilde{v}_1 \in \tilde{V}$ and derive from this definition the sets $\tilde{V}_0$, containing $\tilde{v}_0$ for every $v \in V$, and $\tilde{V}_1$ likewise. Thus, $\tilde{V}_0$ and $\tilde{V}_1$ form a partition of $\tilde{V}$. Furthermore, we introduce a directed edge $\tilde{e}_v = (\tilde{v}_0, \tilde{v}_1) \in \tilde{E}$ for every $v \in V$ with $c(\tilde{e}_v) = w(v)$, where $c\colon \tilde{E} \to \mathbb{R}_{\geq 0}$ defines the capacity, see Figure 2.10. Besides, we replace every edge $f \in E$ with two directed edges $\tilde{e}_0^f, \tilde{e}_1^f \in \tilde{E}$ going in opposite directions to maintain the structure of the graph. With defining $c(\tilde{e}_0^f) = c(\tilde{e}_1^f) = m \in \mathbb{R}$ with $m > \sum_{v \in V} w(v)$ we complete the transformation. We notice that $\tilde{E}|_{\tilde{V}_0 \to \tilde{V}_1}$ is the set of edges induced by V (black edges in Figure 2.10) and $\tilde{E}|_{\tilde{V}_1 \to \tilde{V}_0}$ is the set of edges induced by E (red).

With this set-up, a vertex separator $S \subseteq V \setminus \{s,t\}$ corresponds to an edge set $\tilde{E}_S \coloneqq \left\{ \tilde{e}_v \in \tilde{E} \,\middle|\, v \in S \right\} \subseteq \tilde{E}|_{\tilde{V}_0 \to \tilde{V}_1} \setminus \{\tilde{e}_s, \tilde{e}_t\}$. Using this edge set, we can define a set

$$X \coloneqq \tilde{V} \setminus \left(\left\{ \tilde{v}_1 \in \tilde{V}_1 \,\middle|\, v \in S \right\} \cup \{\tilde{t}_0, \tilde{t}_1\} \right)$$

such that $(X, \tilde{V} \setminus X)$ defines an $\tilde{s}_1$-$\tilde{t}_0$ cut in $\tilde{G}$ with $c(X, \tilde{V} \setminus X) = \sum_{\tilde{e}_v \in \tilde{E}_S} c(\tilde{e}_v) = \sum_{v \in S} w(v)$. Hence, a minimum-weight vertex separator in G can be found by identifying an $\tilde{s}_1$-$\tilde{t}_0$ cut (X, Y) of minimal capacity in $\tilde{G}$ such that

$$\tilde{E}|_{X \to Y} \subseteq \tilde{E}|_{\tilde{V}_0 \to \tilde{V}_1} \setminus \{\tilde{e}_s, \tilde{e}_t\}\,. \tag{2.5}$$

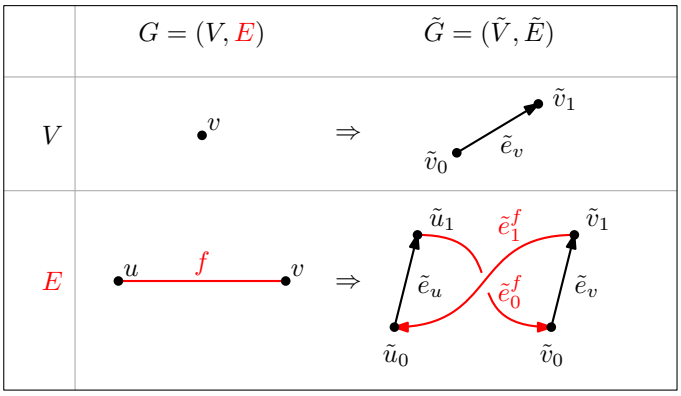


Figure 2.10: Transformation from G to $\tilde{G}$. On the right hand side, elements induced by V are black, those induced by E are red.

For any $\tilde{s}_1$-$\tilde{t}_0$ cut (X, Y) that does not hold Equation 2.5, an edge $\tilde{e} \in \tilde{E}|_{\tilde{V}_1 \to \tilde{V}_0}$ with $c(\tilde{e}) = m$ contributes to $c(X, Y)$. The pair $(\tilde{V}_0, \tilde{V}_1)$ defines an $\tilde{s}_1$-$\tilde{t}_0$ cut with $c(\tilde{V}_0, \tilde{V}_1) = \sum_{\tilde{e} \in \tilde{E}|_{\tilde{V}_0 \to \tilde{V}_1}} c(\tilde{e}) = \sum_{v \in V} w(v) < m$. Thus, a minimum cut in $\tilde{G}$ fulfills Equation 2.5 and, consequently, yields a minimum-weight vertex separator in G, see Figure 2.11.

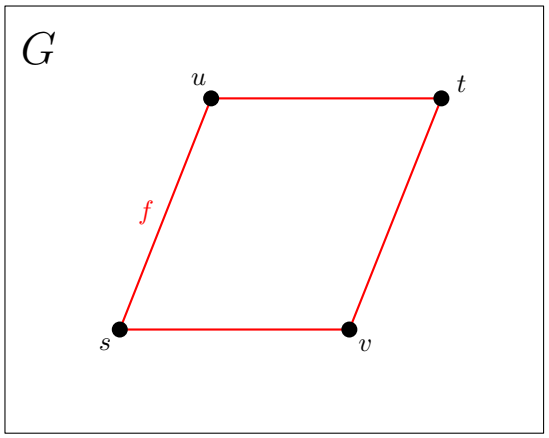
2.3 Linear programming and (mixed-)integer linear programming

Some of the optimization problems that are tackled in the following chapters can be formulated as *linear-programming problems* or *(mixed-)integer linear-programming problems*. Thus, it is possible to apply the well-founded theory of *mathematical optimization* to these problems. Moreover, we can use sophisticated tools that are based on this theory. In Section 2.3.1, we give an introduction to linear programming as it is applied to the problem in Chapter 5. Also, established procedures for solving (mixed-)integer linear programs rely on linear programming. We present them in Section 2.3.2 since we formulate the problem of Chapter 4 as a mixed-integer linear programming problem. For a detailed presentation and analysis, we refer to Nemhauser and Wolsey [NW88]. This introduction is also based on their textbook.

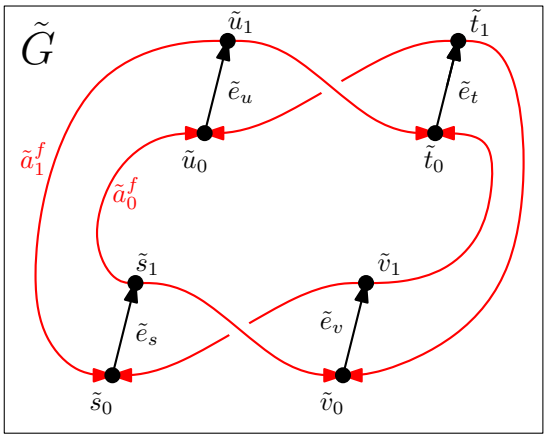
2.3.1 Linear programming

Problem definition Linear programming deals with linear optimization problems, i.e. linear *minimization* and *maximization problems*, that are bounded by linear constraints. A linear program (LP) is composed of

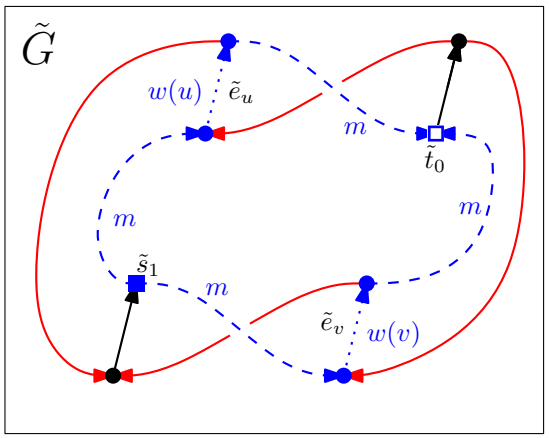
- *variables*, commonly denoted as a vector $x \in \mathbb{R}^n_{\geq 0}$ of n non-negative unknowns,
- *objective* or *cost function*, a linear expression in x, commonly denoted as the product of a vector $c \in \mathbb{Z}^n$ and x,
- *constraints*, a set of m restrictions that are expressed linearly in x, commonly denoted with a matrix $A \in \mathbb{Z}^{m \times n}$ and a vector $b \in \mathbb{Z}^m$ such that $Ax \leq b$ holds.



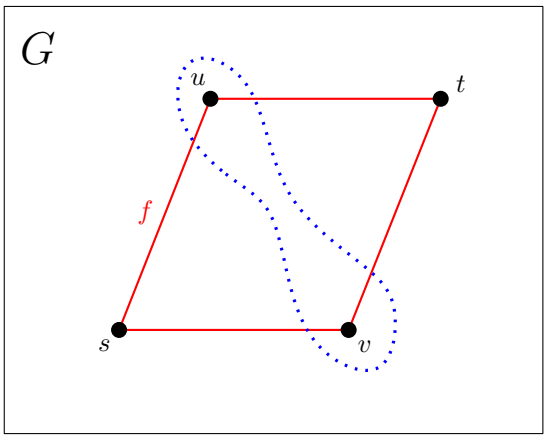
(a) A quadrilateral as an explanatory graph $G = (V, E)$. Here, we are looking for a minimum-weight (s, t)-separator.



(b) The transformation $\tilde{G} = (\tilde{V}, \tilde{E})$ of G; elements in $\tilde{V}$ and $\tilde{E}$ induced by V are black, those in $\tilde{E}$ induced by E are red.



(c) Blue edges are usable for a flow from $\tilde{s}_1$ (■) to $\tilde{t}_0$ (□) and have their capacities annotated. Since m is a sufficiently large constant, the maximum flow is $w(u) + w(v)$, limited by directed edges induced by V (here: the dotted edges $\tilde{a}_u$, $\tilde{a}_v$ with weights $w(u)$, $w(v)$ respectively).



(d) Figure (c) finally provides a minimum-weight edge cut $\{\tilde{a}_u, \tilde{a}_v\}$. Transferring this result back to G, we get the sought minimum-weight (s, t)-separator $\{u, v\}$.

Figure 2.11: Finding a minimum-weight vertex separator.

The objective of linear programming is to find values for x that maximize (or minimize) the objective function and satisfy each constraint. As a decision problem, we can define linear programming as follows.

	LinearProgramming
Instance:	A matrix $A \in \mathbb{Z}^{m \times n}$, bounds $b \in \mathbb{Z}^m$
	cost vector $c \in \mathbb{Z}^n$
	an integer $v \in \mathbb{Z}$
Question:	Is there a vector $x \in \mathbb{R}^n$ such that $Ax \leq b$ and $cx \geq v$?

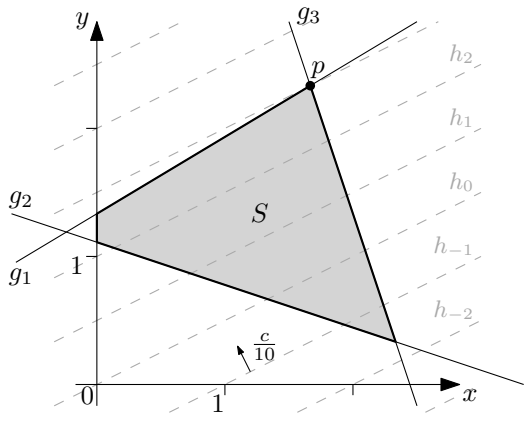
In *canonical form*, an LP then describes a problem

$$\begin{aligned} \max \quad cx \quad & \text{such that} \\ Ax &\leq b \\ x &\geq 0. \end{aligned} \tag{2.6}$$

This problem formulation also includes minimization problems as minimizing cx is equivalent to maximizing $-cx$. Also, constraints are not limited to terms with an upper bound. Constraints with lower bounds can easily be expressed as upper-bound constraints by multiplication with -1. Equality constraints $A'x = b'$ can be expressed with two inequalities $A'x \leq b'$ and $-A'x \leq -b'$. However, when applying linear programming to problems, it is common to go without the canonical form. In this case, constraints are often written line for line with "$\leq$", "$\geq$", or "$=$" to increase readability.

Interpreting the problem geometrically, see Figure 2.12 in Example 2.2, each of the constraints describes a half-space in $\mathbb{R}^n$. Consequently, all constraints as a whole describe the intersection of these half-spaces, a (possibly empty or unbounded) *polytope* S. Any $x \in S$ holds every inequality of the constraint set and, thus, presents a *feasible solution* to the LP. The polytope S is called *feasible region*. The LP itself is *feasible* if $S \neq \emptyset$. The objective function cx defines a family of parallel hyperplanes $h_v = \{x \in \mathbb{R}^n | \, cx = v\}$ where v is the *objective value* of x. In a maximization problem, the maximum $v \in \mathbb{R}$ such that $h_v \cap S \neq \emptyset$ defines the *maximal value* of the solution. Analogously, the *minimal* or, in general, *optimum value* is defined. For the optimal value v, a feasible solution x in h_v, i.e. an $x \in h_v \cap S$, is called an *optimal solution*. If there is no optimal value, the polytope and, thus, the LP is *unbounded*.

Example 2.2. *Consider the following two-dimensional linear program:*



$$\begin{array}{lll} \max & -x+2y & \textit{such that} \\ & & -9x+15y \leq 20 \\ & & 2x+\ 7y \geq 7 \\ & & 9x+\ 3y \leq 22 \\ & & x, y \geq 0 \\ & & x, y \in \mathbb{R} \end{array} \tag{2.7}$$

Figure 2.12: Geometrical representation of the LP in Equation 2.7.

Interpreting the constraints geometrically, we get a two-dimensional polytope, i.e. a polygon S. This polygon is depicted in Figure 2.12. It is bounded by the lines g_1, g_2, and g_3, corresponding to the constraints in the same order, as well as the axes due to $x, y \geq 0$. The objective function is defined with a vector c. A selection of the parallel, two-dimensional hyperplanes of equal value, i.e. the lines h_v, is depicted as gray, dashed lines.
Maximizing the LP in Equation 2.7 can be solved graphically by pushing h_v along c as long as $h_v \cap S \neq \emptyset$. Apparently, this condition is fulfilled until $p = (5/3, 7/3)$ with $g_1 \cap g_3 = \{p\}$ is reached. The point p lies on h_3 since $-5/3 + 2 \cdot 7/3 = 3$.

Solution strategies and complexity The fact that in Example 2.2 an optimal solution is found in an extreme point is no coincidence. Actually, if there is an optimal solution to an LP, then there is always one in an extreme point of the corresponding polytope [NW88]. This fact is the foundation for the most common algorithm for solving LPs, the *simplex algorithm*. The simplex algorithm explores the polytope of an LP from extreme point to extreme point. As soon as no further improvement of the objective value is possible, the algorithm halts. For a detailed presentation of this algorithm and, in particular, reasons why it is easy to detect that no further improvement is possible, we refer to Nemhauser and Wolsey [NW88]. It has been shown that the simplex algorithm has an exponential worst-case running time. Nevertheless, according to Nemhauser and Wolsey, it is used in all commercial linear programming tools since it deals successfully with real-world problems. In fact, probabilistic studies have shown that under general assumptions, a polynomial running time can be expected [ST04]. However, LinearProgramming $\in \mathbf{P}$ since polynomial time algorithms like the *ellipsoid algorithm* or Karmarkar's algorithm exist. The latter is the first *interior-point method* (in contrast to the simplex algorithm, which explores the boundary) and "often competitive with the simplex method" [CLR90].

2.3.2 (Mixed-)Integer linear programming

In Chapter 4, we use linear optimization tools on a combinatorial, i.e., discrete problem. A way of modeling such a problem is by demanding some or even all variables to be *inte-*

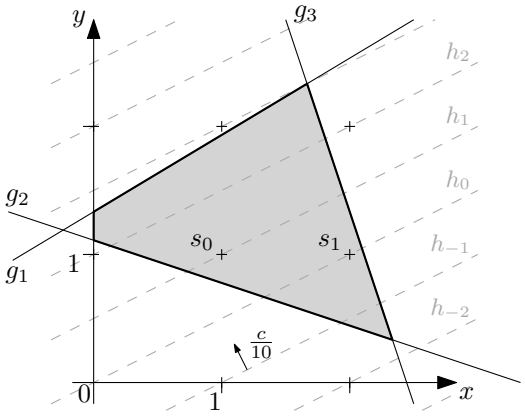
gral. Thus, MixedIntegerLinearProgramming and IntegerLinearProgramming are defined as follows:

	MixedIntegerLinearProgramming
Instance:	Matrices $A, A' \in \mathbb{Z}^{m \times n}$, bounds $b \in \mathbb{Z}^m$
	cost vector $c, c' \in \mathbb{Z}^n$
	an integer $v \in \mathbb{Z}$
Question:	Are there vectors $x \in \mathbb{Z}^n$, $x' \in \mathbb{R}^n$ such that $Ax + A'x' \leq b$ and $cx + c'x' \geq v$?

	IntegerLinearProgramming
Instance:	A matrix $A \in \mathbb{Z}^{m \times n}$, bounds $b \in \mathbb{Z}^m$
	cost vector $c \in \mathbb{Z}^n$
	an integer $v \in \mathbb{Z}$
Question:	Is there a vector $x \in \mathbb{Z}^n$ such that $Ax \leq b$ and $cx \geq v$?

Apparently, the three problems in Sections 2.3.1 and 2.3.2 are closely related. IntegerLinearProgramming and LinearProgramming are both special cases of MixedIntegerLinearProgramming with $A' = 0$ and $A = 0$, respectively.

Example 2.3. *Consider the following MILP:*



$$\begin{array}{lrr} \max & -x + 2y & \textit{such that} \\ & -9x + 15y & \leq 20 \\ & 2x + 7y & \geq 7 \\ & 9x + 3y & \leq 22 \\ & x, y & \geq 0 \\ & x, y & \in \mathbb{Z} \end{array} \tag{2.8}$$

Figure 2.13: Geometrical representation of the MILP in Equation 2.8.

The linear constraints define the same polytope as in Example 2.2. However, this time, the feasible region S is constrained to $\mathbb{Z}$ and, consequently, $S = \{s_0, s_1\} = \{(1,1),(1,2)\}$. Apparently, s_0 solves the problem optimally with an optimal value of 1*. It can be seen that*

the solution s_0 to the MILP is not directly related to the solution $p = (5/3, 7/3)$ to the LP in Example 2.2; neither rounding p nor choosing the closest feasible solution to p does yield s_0.

Since the solution to an MILP does not necessarily represent an extreme point of the polytope (see Figure 2.13 in Example 2.3), simply applying the simplex algorithm does not yield an optimal solution. In fact, MixedIntegerLinearProgramming $\in$ **NP**-complete [GJ90] and, thus, no efficient algorithm is known nor expected to be designed.

Problem relaxation and cutting planes A general approach to solving MILPs begins with a *relaxed* problem, i.e. a problem with a bigger feasible region containing the feasible region S of the MILP. This relaxed problem is iteratively solved and constrained until its optimal solution is an element of S and, thus, an optimal solution to the MILP. This procedure is summarized in Algorithm 10.

Algorithm 10: General-Relaxation-Algorithm(MILP M)

Data: MILP M with feasible region S
Result: optimal solution x for M

```
1 i ← 0;
2 relax M with S to MILP M_0 with S_0 ⊇ S;
3 while true do
4     x ← solve(M');
5     if x ∈ S then
6         return x;
7     constrain M_i to M_{i+1} such that S ⊆ S_{i+1} ⊆ S_i \ {x};
8     i ← i + 1;
```

A common approach for a relaxed MILP in Line 2 in Algorithm 10 is to relax the integrality constraint of the MILP and start with the *LP relaxation*. For any mixed integer linear program (MILP)

$$\begin{aligned} \max \quad & cx + c'x' && \text{such that} \\ & && Ax + A'x' \leq b \\ & && x \geq 0 \\ & && x' \geq 0 \\ & && x \in \mathbb{Z}^n \\ & && x' \in \mathbb{R}^n \end{aligned} \tag{2.9}$$

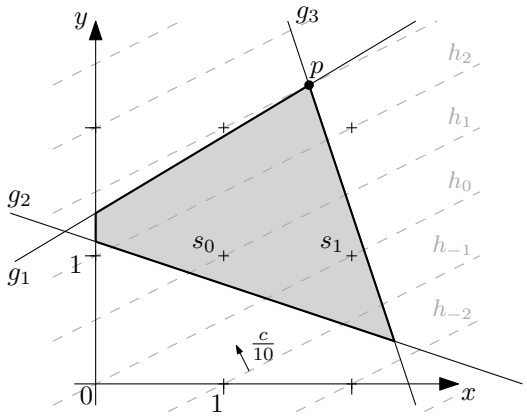
the LP relaxation is defined as

$$\begin{aligned} \max \quad & cx + c'x' && \text{such that} \\ & && Ax + A'x' \leq b \\ & && x \geq 0 \\ & && x' \geq 0 \\ & && x \in \mathbb{R}^n \\ & && x' \in \mathbb{R}^n. \end{aligned} \tag{2.10}$$

In the following, for an MILP with a feasible set S, the feasible set of the corresponding LP relaxation is denoted as S_{LP}.

An implementation of Algorithm 10 for MILPs starts with the LP relaxation. The restrictions in Line 7 are realized with additional linear constraints. These constraints are called *cutting planes* since they cut off parts of the polytope of the LP relaxation. In order to preserve the restriction in Line 7, in the LP relaxation, it is important that only non-integral solutions are cut off from the polytope.

Example 2.3 (continued)**.** *Carefully chosen, just two cutting planes suffice such that the LP relaxation yields s_0 as an optimal solution.*



$$\begin{array}{lrl} \max & -x + 2y & \textit{such that} \\ & -9x + 15y & \leq 20 \\ & 2x + 7y & \geq 7 \\ & 9x + 3y & \leq 22 \\ & -x + y & \leq 0 \\ & & y \leq 1 \\ & & x, y \geq 0 \\ & & x, y \in \mathbb{Z} \end{array} \tag{2.11}$$

Figure 2.14: Geometrical representation of the MILP in Equation 2.8 and the corresponding LP relaxation...

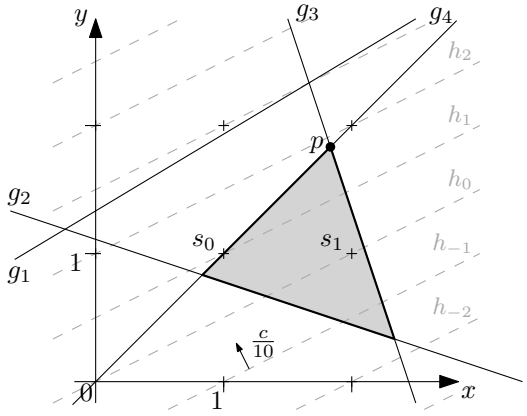


Figure 2.15: ... with $-x + y \leq 0$ (g_4)...

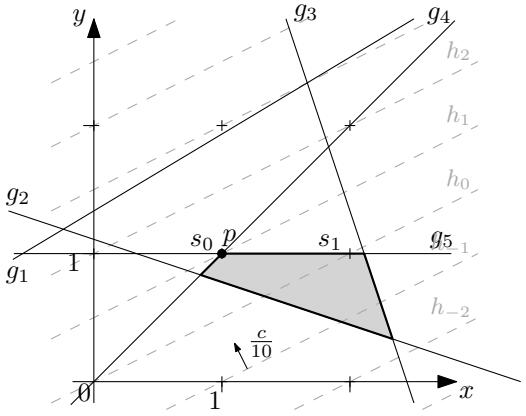


Figure 2.16: ... and $y \leq 1$ (g_5).

Adding $-x + y \leq 0$, the optimal solution p to the LP relaxation and s_0 do not coincide yet, see Figure 2.15. Only with the additional cut $y \leq 1$, they are identical, i.e. $p = s_0$, see Figure 2.16.

However, choosing cutting planes carefully is seldom as easy as in this example. A general approach to finding cutting planes are for example *Gomory cuts*. Sometimes problem specific

cutting planes speed up the solving process. This is the case in Chapter 4, which is why we refer to Nemhauser and Wolsey [NW88] for a detailed presentation of general cutting-plane algorithms.

Branch-and-bound Another established procedure for solving MILPs is a *divide-and-conquer* approach. This is an algorithm-design paradigm that is based on the idea of splitting hard problems into smaller instances that are easier to solve. Then, the solutions of these instances are combined to retrieve a solution to the original problem. This concept is applicable to MixedIntegerLinearProgramming. The *branch-and-bound* procedure follows this paradigm. Here, the feasible region S is divided into a set $\{S_i\}_{i=0}^{k}$ such that $\bigcup_{i=0}^{k} S_i = S$. Due to

$$\max\{cx|\, x \in S\} = \max_{i=0}^{k}(\max\{cx|\, x \in S_i\})$$

for any division of S, combining found solutions to the subproblems in order to retrieve an optimal solution to the original problem is not an issue.

The starting point of any branch-and-bound algorithm is an MILP which is hard to solve. Also, easy-to-find relaxations of this MILP do not yield solutions that are feasible with respect to the original problem. The problem is then split into two or more subproblems and the procedure is applied recursively. This part of the procedure is known as *branching*. During the running time, the feasible solution with the currently best objective value is kept in the memory; modern solvers refer to it as the *incumbent solution* [CPL17, Gur18]. This incumbent solution gives a bound for the objective value of an optimal solution. A new incumbent solution initiates the *bounding* step. Subproblems that do not yield solutions better than the incumbent one do no longer play a role in the search for an optimal solution to the original problem. These subproblems get pruned from the *branch-and-bound tree*.

In the following, we will focus on branch-and-bound algorithms that work with LP relaxations as they are most common in modern-day solvers, compare Algorithm 11. Here, branching is done with respect to violations concerning the integrality of a variable x of the MILP. Let x' be the value of x in the solution to the LP relaxation and $\lfloor x' \rfloor$ the largest integer value smaller than x', i.e. $\lfloor x' \rfloor \coloneqq \max\{i \in \mathbb{Z}|\, i < x'\}$. Adding $x \leq \lfloor x' \rfloor$ and $x \geq \lfloor x' \rfloor + 1$, respectively, produces two subproblems excluding solutions with $x = x'$. Each of these subproblems is the root of a subtree of the branch-and-bound tree. The tree of a subproblem M' of an MILP M can be pruned if one of the following conditions is met:

(P1) M' is infeasible.

(P2) The solution x_{LP} to M'_{LP} is feasible for M, i.e. $x_{\text{LP}} \in \mathbb{Z}^n$.

(P3) With respect to the objective, the solution x_{LP} to M'_{LP} is not better than the current incumbent solution x, i.e., in the case of a maximization problem, $c \cdot x_{\text{LP}} \leq c \cdot x$ holds.

Algorithm 11 offers possibilities for variations. Line 5 does not specify the order in which the MILPs are explored in the branch-and-bound tree. However, commonly, a depth-first search is preferred over a breadth-first search. Also, Line 13 requires good strategies for choosing the variable that is used for the branching step.

Algorithm 11: LP-Relaxation-Branch-and-Bound-Algorithm(MILP M)

Data: MILP M with feasible region S
Result: optimal solution x_{opt} for M

$\mathscr{P} \leftarrow \{M\}$;
// setting lower and upper bound
$l \leftarrow -\infty$, $x_{\text{opt}} \leftarrow$ NIL;
while $\mathscr{P} \neq \emptyset$ **do**
    extract MILP M' from $\mathscr{P}$ and let S' be its feasible set;
    solve M'_{LP} and define optimal value z_{LP} and optimal solution x_{LP} if existing;
    if $z_{\text{LP}} > l$ **then**
        if $x_{\text{LP}} \in S'$ **then**
            $l \leftarrow z_{\text{LP}}$;
            $x_{\text{opt}} \leftarrow x_{\text{LP}}$;
        else
            // enforce integrality on integer variable with non-integral value
            extract i from $\left\{j \in \{1, \ldots, n\} \,\middle|\, x'_j \notin \mathbb{Z}\right\}$;
            complete M' to M'_1 with $x_i \leq \lfloor x'_i \rfloor$ and to M'_2 with $x_i \geq \lfloor x'_i \rfloor + 1$, respectively;
            $\mathscr{P}$.add($\{M'_1, M'_2\}$);
return x_{opt};

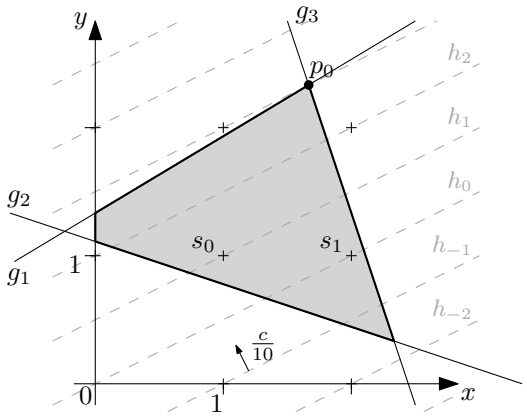
Example 2.3 (continued)**.** *In the following, we use branch-and-bound to solve the following MILP $M^{(0)}$ with feasible set S_0 (described first in Equation 2.8):*

$$\begin{array}{lrr} \max & -x + 2y & \textit{such that} \\ & -9x + 15y & \leq 20 \\ & 2x + 7y & \geq 7 \\ & 9x + 3y & \leq 22 \\ & x, y & \geq 0 \\ & x, y & \in \mathbb{Z} \end{array}$$

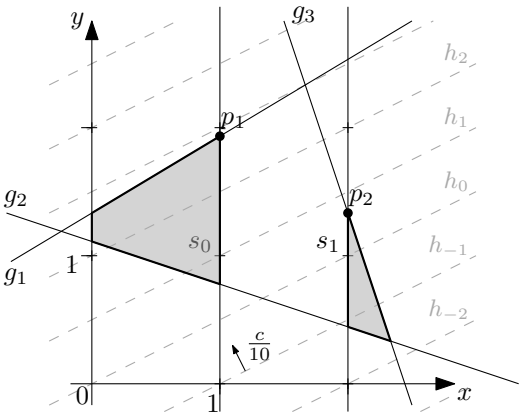
Figure 2.17(e) depicts the corresponding branch-and-bound tree.

The LP relaxation $M_{\text{LP}}^{(0)}$ to this problem yields $p_0 = (5/3, 7/3)$ as an optimal solution, see Figure 2.17(a). Since $p_0 \notin \mathbb{Z}^2$, further steps are necessary to find an integral solution. Branching on x yields the constraints $x \leq 1$ and $x \geq 2$ which, added to the original problem respectively, produce the two subproblems $M^{(1)}$ and $M^{(2)}$ with $S_1 = S_0 \cap \{(x, y) \in \mathbb{R}^2 \mid x \leq 1\}$ and $S_2 = S_0 \cap \{(x, y) \in \mathbb{R}^2 \mid x \geq 2\}$, respectively. Their LP relaxations $M_{\text{LP}}^{(1)}$ and $M_{\text{LP}}^{(2)}$ with their optimal solutions are depicted in Figure 2.17(b). The optimal solution p_1 to $M_{\text{LP}}^{(1)}$ is not integral, either. Consequently, another branching step is necessary, this time on y. The constraints $y \leq 1$ and $y \geq 2$ again produce two new subproblems $M^{(3)}$ and $M^{(4)}$ with $S_3 = S_1 \cap \{(x, y) \in \mathbb{R}^2 \mid y \leq 1\}$ and $S_4 = S_1 \cap \{(x, y) \in \mathbb{R}^2 \mid y \geq 2\}$, respectively, see Figure 2.17(c). Apparently, $S_4 = \emptyset$. Thus, the problem $M^{(4)}$ is infeasible and no further branching is necessary according to pruning criterion (P1). The optimal solution p_3 to $M^{(3)}$ is not integral. Branching $M^{(3)}$ on x yields $M^{(7)}$ and $M^{(8)}$ with $S_7 = S_3 \cap \{(x, y) \in \mathbb{R}^2 \mid x \leq 0\}$ and $S_8 = S_3 \cap \{(x, y) \in \mathbb{R}^2 \mid x \geq 1\}$, respectively, see Figure 2.17(d). For $M^{(7)}$ the feasible region is empty and the branching stops according to (P1). Solving $M_{\text{LP}}^{(8)}$ yields p_8 as an optimal solution. Since $p_8 \in \mathbb{Z}^2$, it also solves $M^{(8)}$ optimally. The solution p_8 is now the incumbent solution to $M^{(0)}$ with an objective value of 1. The branching procedure terminates according to pruning criterion (P2). Now we turn to the subproblems of $M^{(2)}$. Figure 2.17(b) indicates that p_2 is an optimal solution to $M_{\text{LP}}^{(2)}$. Since $c \cdot p_2 = 2/3 < 1 = c \cdot p_8$, there is no need to explore this branch any further according to (P3). A feasible solution to $M^{(0)}$ in this branch has an objective value of less than 1. Nevertheless, in this example, we continue with the branching procedure and face the problems $M^{(5)}$ and $M^{(6)}$ with $S_5 = S_2 \cap \{(x, y) \in \mathbb{R}^2 \mid y \leq 1\}$ and $S_6 = S_2 \cap \{(x, y) \in \mathbb{R}^2 \mid y \geq 2\}$, respectively, see Figure 2.17(c). The solution p_5 to $M_{\text{LP}}^{(5)}$ is integral and, thus, an optimal solution to $M^{(5)}$. The feasible region of $M^{(6)}$ is empty. In both cases, branching terminates due to (P1) and (P2), respectively.

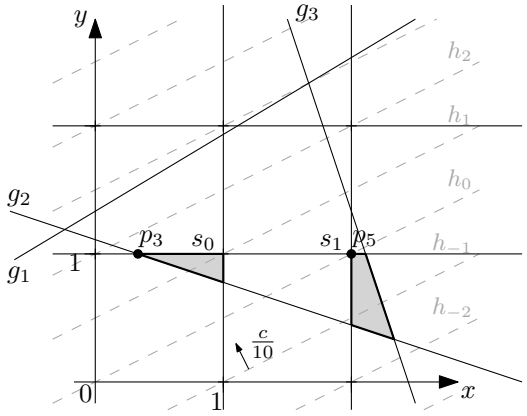
Branch-and-cut Both approaches described above, cutting-plane and branch-and-bound algorithms, can be applied in combination. Adding cutting planes to a subproblem M', i.e. a node in the branch-and-bound tree, may decrease the number of necessary branching steps in the following. The cutting planes applied can be valid locally, i.e. only in the subtree with the root M', or globally if they define valid cuts for the original problem. This combined approach is commonly referred to as *branch-and-cut* [PR91, Mit02].



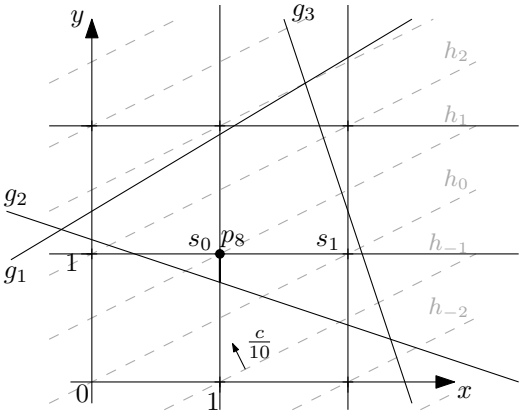
(a) LP relaxation with optimal solution $p_0 \notin \mathbb{Z}^2$.



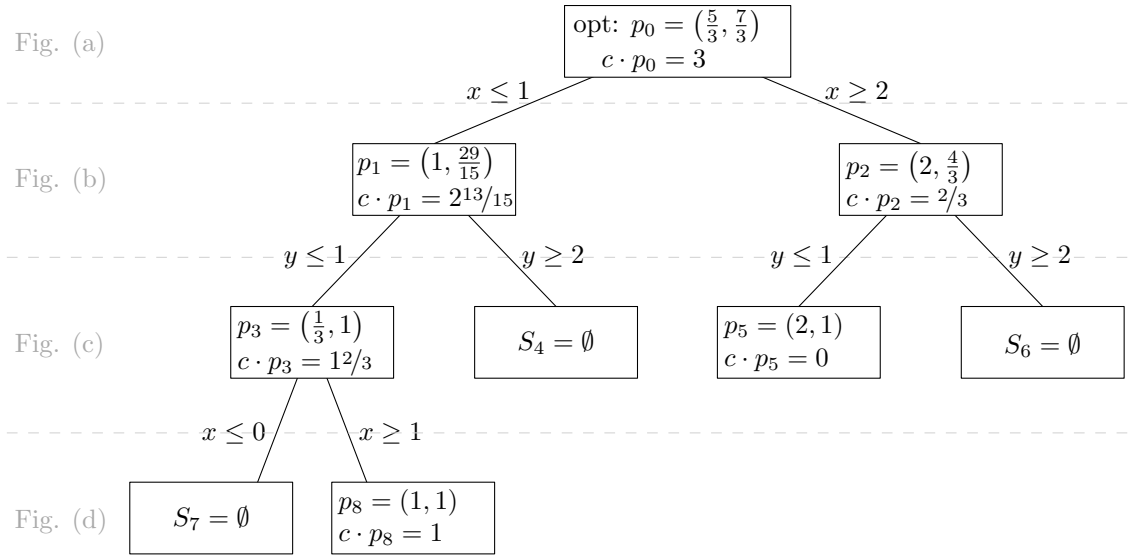
(b) Branching on x yields two subproblems.



(c) Both subproblems in Fig. (b) suggest branching with $y \leq 1$ and $y \geq 2$. For $y \geq 2$, both resulting problems are infeasible; $p_5 \in \mathbb{Z}^2$ holds.



(d) The solution p_3 in Fig. (c) suggests branching on x another time. The feasibility set S_8 is only a line with extreme point p_8.



(e) Branch-and-bound tree describing the procedure depicted in Figures (a) – (d). Here, in the leaves, either an integral solution is found or the feasible set is empty.

Figure 2.17: Exemplary application of branch and bound.

Example 2.4. *Now, we want to solve* MaximumFlow *as presented in Section 2.2.3 with a linear programming approach. Given a directed graph $G = (V, E)$ with distinguished vertices $s, t \in V$ and a capacity $c\colon E \to \mathbb{R}_{\geq 0}$, we are looking for the maximum flow from s to t. The flow from s to t is composed of the flows on the edges of G. Hence, we introduce a variable $f_{(u,v)}$ for every edge $(u, v) \in E$ that represents the flow on this edge in the solution.*

We model the maximization of the flow by maximizing the outflow of s, see the objective in Equation 2.12, and prohibiting the inflow into s, see Equation 2.13. Expressing the capacity constraint and the flow conservation with linear constraints completes the LP. The capacity constraint, i.e. that for every edge $(u, v) \in E$ the flow is bounded by the edge's capacity, is implemented in Inequation 2.14. In order to conserve the flow in every vertex $v \in V \setminus \{s, t\}$, we introduce the constraints in Equation 2.15.

$$\max \sum_{(s,v)\in E} f_{(s,v)} \qquad \text{such that} \tag{2.12}$$

$$\sum_{(u,s)\in E} f_{(u,s)} = 0 \tag{2.13}$$

$$f_{(u,v)} \leq c\,(u, v) \qquad \text{for each } (u, v) \in E \tag{2.14}$$

$$\sum_{(u,v)\in E} f_{(u,v)} - \sum_{(v,w)\in E} f_{(v,w)} = 0 \qquad \text{for each } v \in V \setminus \{s, t\} \tag{2.15}$$

$$f_{(u,v)} \geq 0 \qquad \text{for each } (u, v) \in E \tag{2.16}$$

Since no inflow into s is allowed and the flow is conserved in any vertex but s and t, each unit of commodity entering the network via s has to leave the network via t. Hence, maximizing the outflow from s, we maximize the s-t flow and, thus, solve MaximumFlow.

3 Location-dependent generalization of road networks based on equivalent destinations

The following chapter is mainly taken from a joint work with Thomas C. van Dijk and Jan-Henrik Haunert [vDHO16]. Here, we present an efficient algorithm for the computation of a primarily geometric generalization of a road network. For this purpose, a distinguished location within the road network, the *source*, is considered. Starting from the source, shortest paths to any location within the network are considered. Parts of the road network (both locations and road segments) which are reachable via similar shortest paths get simplified to a point feature. This procedure can be called both a *selection* and an *aggregation* process, compare Figure 1.1. For the visualization, we use only geometric objects that already existed in the input. Hence, based on the input, we select point features and road segments connecting them with the source. However, these selected point features represent parts of the input road network. That means, said parts of the road network get aggregated to point features.

Abstract
Suppose a user located at a certain vertex in a road network wants to plan a route using a wayfinding map. The user's exact destination may be irrelevant for planning most of the route, because many destinations will be equivalent in the sense that they allow the user to choose almost the same paths. We propose a method to find such groups of destinations automatically and to contract the resulting clusters in a detailed map to achieve a simplified visualization. We model the problem as a clustering problem in rooted, edge-weighted trees. Two vertices are allowed to be in the same cluster if and only if they share at least a given fraction of their path to the root. We analyze some properties of these clusterings and give a linear-time algorithm to compute the minimum-cardinality clustering. This algorithm may have various other applications in network visualization and graph drawing, but in this paper we apply it specifically to the generalization of focus-and-context maps, i.e., maps that bring out certain parts of their content in detail and also display the surroundings for increased readability. When contracting shortest-path trees in a geographic network, the computed clustering additionally provides a constant-factor bound on the detour that results from routing using the generalized network instead of the full network. This is a desirable property for wayfinding maps.

3.1 Introduction

Suppose a user wants to get by car from Berlin, Germany to a certain address in Groningen, The Netherlands. She would like to plan her route using a wayfinding map. However, she does not want to be bothered with a detailed map of all of Europe. At the start of her journey, the exact address in Groningen is irrelevant to her, since it would not influence the direction she should start driving in – or, in fact, even where to drive within, say, the next hour. Therefore, the map in a dynamic visualization system may well collapse the city (or even the province) Groningen to a single vertex: any particular destination within Groningen leads to the exact same wayfinding decisions at this point. Only later does the exact address in Groningen become relevant: the map representation of Groningen should change while the user approaches her destination.

In this paper we formally define whether it makes sense to distinguish between a set of destinations. Let the graph $G = (V, E)$ represent the road network and $s \in V$ be the user's current location. Our idea is to compare, for any two vertices $u, v \in V$, the shortest path P_{su} from the user's position s to u, and the shortest path P_{sv} from s to v. If P_{su} *shares* a user-specified fraction $\alpha \in [0, 1]$ of its length with P_{sv} and vice versa, we consider u and v to be equivalent destinations, whose distinction from the user's current perspective is not meaningful. This definition makes sense if α is set to a relatively high number, for example if $\alpha = 0.95$. With this setting, the user can safely navigate for a relatively long distance without having to bother about the distinction between u and v. We consider a set $S \subseteq V$ of destinations equivalent (with respect to a start vertex s and a threshold α) if all vertices in S are mutually equivalent. We will reduce the level of detail of a network by using this equivalence condition to decide which locations can be aggregated.

The reduction of the level of detail of a map is a classical problem in cartography, where it is commonly termed *map generalization* and often approached by optimization [WJT03, Ses05]. Because map generalization has turned out to be a highly complex problem, however, it is typically subdivided into multiple tasks such as the selection of objects for the output map [TP66], the aggregation of objects [HW10a], and the simplification of lines [DDS09]. When generalizing geographic networks, the first step is usually to select edges of a graph representation of the network. This step aims to reduce the visual clutter in the output map while preserving characteristic properties and high-level structures of the network, for example, connectivity [MB93, Zha05] and sequences of line segments that are perceived as groups [TB02]. Often an aim is to preserve vertices or edges of high importance, which can be defined based on graph theoretical centrality measures [JC04]. Brunel et al. [BGK$^+$14] modeled the generalization of networks as optimization problems, showed **NP**-hardness for several of their models, and developed approximation algorithms as well as efficient heuristics. Chimani et al. [CvDH14] considered a problem in which the edges of a graph have to be removed iteratively to produce a sequence of subsequently more generalized maps of a network. They also showed **NP**-hardness for their model and developed efficient approximation algorithms and heuristics.

In this paper, we develop a new model and generalization algorithms based on optimization for focus-and-context network maps, in which some parts of a network are represented with more details than other parts. Such maps have often been suggested for navigation

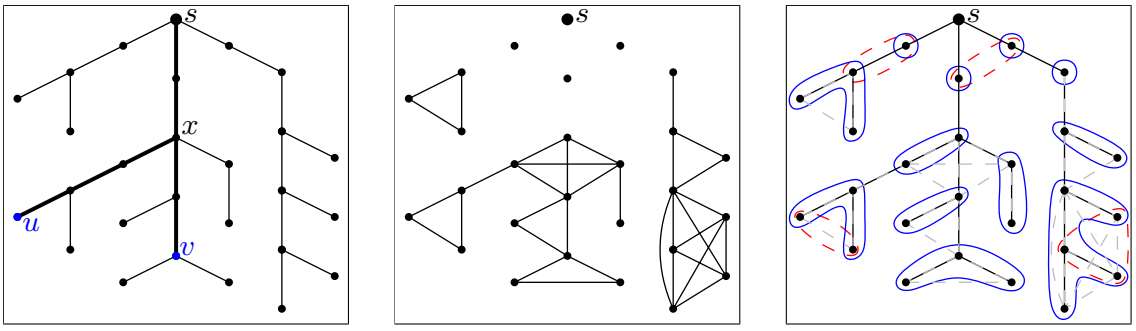
and wayfinding tasks [ZR02], for example, to allow a user to follow a given route [AS01]. Kopf et al. [KAB+10] have presented a method for the automatic generation of *destination maps*, which allow different users to find routes to a given destination. For this purpose, the road network is presented with more details in the destination's vicinity. Focus-and-context maps are often distorted to present highly detailed parts of a network map at a larger scale than other parts. However, we consider the generalization of the network as an interesting problem of its own. After the map has been generalized with our method, different distortion techniques could be applied to use the available space in an optimal way and to reduce the remaining visual clutter, for example, a fish-eye projection [YOT09] or an optimization approach [vDH14]. An overview of distortion techniques for focus-and-context visualization is provided by Cockburn et al. [CKB09].

To summarize, in existing generalization systems, the level of detail is often dictated by a selected map scale. In this paper, however, we develop a method that selects details based on whether they are informative with respect to wayfinding tasks. Unnecessary details are removed and, thus, the visual complexity of the map is greatly reduced. We do not explicitly model graphical conflicts, however, since those may be resolved by selecting an appropriate map scale or using existing distortion techniques.

Our method detects clusters that exist in a road network and in this sense is related to speed-up techniques for shortest-path computations based on contraction hierarchies [GSSV12] or highway hierarchies [SS12]. Such concepts, however, are not readily applicable for the generation of focus-and-context maps of networks.

While this paper primarily considers the generalization of road networks, many other applications also require the visualization of hierarchical information and as such this topic has attracted much attention. Examples include the drawing of business data [VvWvdL06], phylogenetic trees [HRR+07] and file systems [BYB+13]. Particularly the latter two applications typically involve large tree structures where the use of focus-and-context techniques is appropriate if not outright necessary. The concept of focus and context in tree visualization can be traced back at least to Furnas, who proposed fish-eye views where nodes have an importance based on their distance to a focus node [Fur86]. This can be achieved, for example, using hyperbolic trees [LRP95]. These approaches do not explicitly generalize the hierarchical structure, but instead rely on scaling all available node-link information. An alternative drawing style consists of variations of so-called treemaps, for which focused visualizations have also been investigated [BL07].

We now come to the results presented in this paper. We are given a graph $G = (V, E)$ representing the road network, a source vertex $s \in V$, a number $\alpha \in [0, 1]$, and weights $w\colon E \to \mathbb{R}_{\geq 0}$ reflecting edge lengths. In a geographic network, these weights can for example be Euclidean distances, possibly weighted by road class or travel time. Our task is then to partition the vertex set V automatically such that each cell corresponds to a set of equivalent destinations. (Singleton cells are allowed.) Subject to this constraint, we will minimize the number of cells in the partition: this generalizes the road network as much as possible. Since the equivalence condition is based on sharing a *fraction* of a shortest path, clusters near the source s must likely be small and many. Farther away, an optimal solution to the problem will probably have clusters that reflect larger geographic structures.



(a) $\sigma(u,v) = \frac{2}{5}$ and $\sigma(v,u) = \frac{1}{2}$; hence u and v are α-compatible only for $\alpha \leq \frac{2}{5}$.

(b) Compatibility graph $G_\oplus$ for $\alpha = \frac{2}{3}$.

(c) Selection of allowed contractions (blue, computed with Algorithm 12) and counterexamples (red) due to a lack of connectivity in $\mathcal{T}$ or $G_\oplus$.

Figure 3.1: Illustrations for Definitions 3.1 and 3.2, 3.3, and 3.4. All edges have weight 1.

First we formally define this clustering problem in its general setting on arbitrary trees (Section 3.2) and give an efficient linear-time algorithm that solves this problem (Section 3.3). We also observe that a structural theorem plus known algorithms would give another polynomial-time algorithm, but that this is more complicated and less efficient. Then we apply this clustering algorithm specifically to modified shortest-path trees in geographic road networks (Section 3.4). We give several options for visualizing the resulting generalized tree and show how to incorporate cross-tree connectivity information in the drawing. We close with concluding remarks and an outlook on directions for future work (Section 3.5).

3.2 Equivalent destinations in trees

Throughout the paper, we let $G = (V, E)$ be an edge-weighted graph and $\mathcal{T} = (V, E_\mathcal{T})$ a spanning tree of G, rooted at a vertex $s \in V$. Given $\mathcal{T}$, we denote by P_{uv} the unique path in $\mathcal{T}$ that connects u and v; this is well-defined since $\mathcal{T}$ is a tree. Similarly, we write P_{uvw} for the concatenation of P_{uv} and P_{vw}; this notation extends to longer chains of paths. Given a path P, we denote the sum of weights of edges on the path by $w(P)$. Our results hold for arbitrary trees and arbitrary nonnegative edge weights, but in our application we specifically use shortest-path trees and Euclidean distances as weights.

Now we define a measure of similarity between two vertices by how much of their path to s they share: on the basis of this we will decide whether the vertices could reasonably be contracted in a nice visualization. Note that the following definition is not symmetric (see Figure 3.1 for illustrations of the following definitions).

Definition 3.1 (Directed Similarity)**.** *Let $u, v \in V$ be two vertices, $u \neq s$, and let x be their lowest common ancestor in $\mathcal{T}$. The directed similarity of u to v is defined as*

$$\sigma(u,v) = w(P_{sx})/w(P_{su}). \tag{3.1}$$

The directed similarity is always between 0 and 1, inclusive, since paths have nonnegative weight and $w(P_{sx}) \leq w(P_{su})$ since x is on P_{su}.

Definition 3.2 (α-Compatible, $\oplus$)**.** *Two vertices $u, v \in V$ are called α-compatible if and only if both*

$$\sigma(u, v) \geq \alpha \quad \text{and} \quad \sigma(v, u) \geq \alpha. \tag{3.2}$$

When α is clear from context, we write $u \oplus v$ to assert that u and v are compatible.

Note that compatibility is reflexive and symmetric, but not transitive. For the rest of the paper we assume that a constant $\alpha \in [0, 1]$ is used throughout.

Definition 3.3 (Compatibility graph)**.** *The* compatibility graph $G_{\oplus} = (V, E_{\oplus})$ *has an edge between any two vertices $u, v \in V$ if and only if $u \oplus v$.*

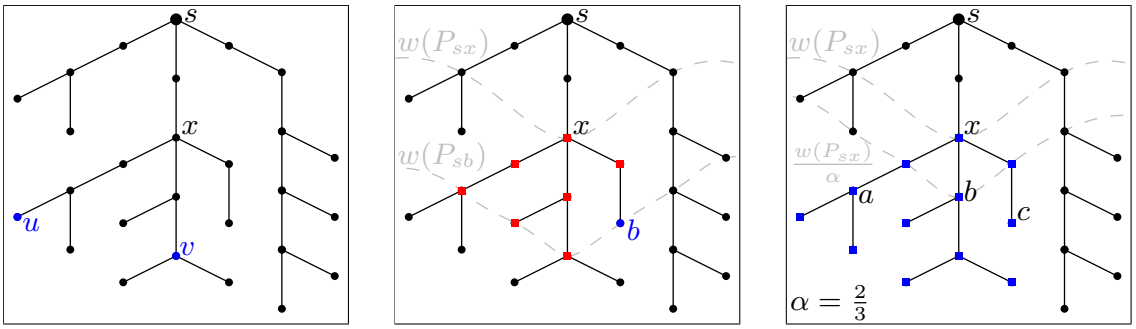
Definition 3.4 (Allowed contraction)**.** *Contracting a set of vertices $S \subseteq V$ is called* allowed *if and only if S is connected in $\mathcal{T}$ and all vertices in S are pairwise α-compatible. Note that the latter is equivalent to S being a clique in $G_{\oplus}$.*

These definitions lead naturally to the following problem statement.

	TreeSummary
Instance:	A tree $\mathcal{T} = (V, E_{\mathcal{T}})$ rooted at $s \in V$.
	Weights on the edges $w : E_{\mathcal{T}} \to \mathbb{R}_{\geq 0}$.
	A compatibility threshold $\alpha \in [0, 1]$.
	An integer k.
Question:	Does there exist a partition of V into at most k cells, such that each cell is an allowed contraction?

Since allowed contractions correspond to cliques, this problem asks for a minimum-cardinality clique cover of $G_{\oplus}$. (The proof of Theorem 3.1 will show that there exists an optimal clique cover where each cell is connected in $\mathcal{T}$.) The CliqueCover problem on general graphs is one of Karp's original 21 **NP**-complete problems [Kar72], but polynomial-time solvable for several restricted graph classes. (Sometimes the term CliquePartition is used instead – interchangeably, it seems. In our situation we cover the vertices, not the edges.) In particular, we solve the optimization version of TreeSummary in linear time. First we give some structural lemmata. Consult Figure 3.2 for illustrations of the involved vertices and sets.

Lemma 3.1 (Compatible descendants)**.** *Let $u, v \in V$ be vertices and let x be their lowest common ancestor. Then u and v are compatible if and only if they are each compatible to x, that is, $u \oplus v \iff u \oplus x \ \wedge \ v \oplus x$.*



(a) The vertices u and v are compatible if and only if they are both compatible to x.

(b) If $b \oplus x$, then all square vertices (red) are compatible with x.

(c) In Lemma 3.3, S must be an appropriate subset of the square vertices, e.g. $S = \{a, b, c, x\}$; then $\{b, x\}$ will form $S_\oplus$, all others, i.e. a and c, are incompatible.

Figure 3.2: Illustrations for Lemmata 3.1–3.3

Proof. Implication both ways.

($\Longrightarrow$) Note that $\sigma(u,x) = \sigma(u,v)$ since x is the lowest common ancestor of u and v. By $u \oplus v$ we have $\sigma(u,v) \geq \alpha$ and therefore $\sigma(u,x) \geq \alpha$. The lowest common ancestor of u and x is in fact x, since u is a descendant of x: then $\sigma(x,u) = w(P_{sx})/w(P_{sx}) = 1 \geq \alpha$. This gives $u \oplus x$. The same argument for v gives $v \oplus x$.

($\Longleftarrow$) Since $u \oplus x$, we have by definition that $\sigma(u,x) \geq \alpha$. Symmetrically $v \oplus x$ gives $\sigma(v,x) \geq \alpha$. Because x is the lowest common ancestor of u and v, this is precisely the definition of $u \oplus v$.

This concludes the proof. □

Lemma 3.2 (Compatible subtrees)**.** *Let $x \in V$ and let $a, b \in V$ be descendants of x. If $w(P_{sa}) \leq w(P_{sb})$ and $b \oplus x$, then $a \oplus x$. That is, if a and b are both descendants of x, and b is farther away from the root, then $b \oplus x \implies a \oplus x$.*

Proof. The lowest common ancestor of b and x is x itself, and we have $b \oplus x$, hence $w(P_{sx})/w(P_{sb}) \geq \alpha$. Since the lowest common ancestor of a and x is x as well, we have $\sigma(a,x) = w(P_{sx})/w(P_{sa}) \geq w(P_{sx})/w(P_{sb}) \geq \alpha$. With $\sigma(x,a) = 1 \geq \alpha$ we get $a \oplus x$. □

Lemma 3.3 (Equivalent descendants)**.** *Let $x \in V$ be a vertex and let $S \subseteq V$ be a set of vertices such that for any pair of vertices in S their lowest common ancestor is x. Then $(\oplus)$ is an equivalence relation on S. In particular, let $S_\oplus = \{v \in S \mid v \oplus x\}$. All pairs of vertices in $S_\oplus$ are compatible, and any vertex in $S \setminus S_\oplus$ is not compatible to any other vertex in S.*

Proof. Directly from Lemma 3.1. Let $u, v \in S$. If $u \oplus x$ and $v \oplus x$, then $u \oplus v$. If either $u \not\oplus x$ or $v \not\oplus x$, then $u \not\oplus v$. □

3.3 A linear-time algorithm for TreeSummary

We solve the optimization version TreeSummary: partitioning V into as few cells as possible, where every cell is an allowed contraction. The algorithm starts with every leaf vertex in a cell of its own and greedily merges cells in a post-order traversal (that is, bottom-up). During the algorithm, every cell C is an allowed contraction, that is, the following invariants hold:

(1) the vertices in C are connected in $\mathcal{T}$ and

(2) C is a clique in $G_{\oplus}$.

When the algorithm is done, the cells partition V and we will prove that the number of cells is minimized.

For a cell C, its vertex with the shortest path to s is called $\mathit{Root}(C)$. A vertex in C with the longest path to s (or tied for the longest) is called *deep*. Throughout the algorithm, we maintain the following references. Every vertex v, for as long as it is the root of a cell C, stores a reference $\mathit{Cell}(v) = C$. This is uniquely defined since cells are disjoint. Every cell C stores a reference $\mathit{Deep}(C)$ to any one of its deep vertices. These references can easily be kept up to date during the algorithm.

The algorithm traverses $\mathcal{T}$ in post-order. After processing a vertex, that vertex is the root of a cell. Therefore, when considering any vertex x, each child $c \in \mathit{Children}(x)$ is the root of a cell. We call a child c contractible if and only if $x \oplus \mathit{Deep}(\mathit{Cell}(c))$. We merge the cells of all contractible children – that is, we take their union – and add x to this cell: this results in a cell with x as its root. We do not do anything with incontractible children; their cells will not change anymore. See Algorithm 12.

This algorithm satisfies the two stated invariants. Upon initialization, both clearly hold. When processing a vertex, Lemma 3.3 holds for the deep vertices of its children's cells and then Lemma 3.2 holds for the entire cell. This shows that the (single) newly constructed cell is an allowed contraction.

Theorem 3.1. *Algorithm 12 computes an optimal solution and runs in $\mathcal{O}(n)$ time, where n is the number of vertices in $\mathcal{T}$.*

Proof. Let $\mathcal{C}$ be the set of cells determined by the algorithm. Note that $\mathcal{C}$ is a clique partition of $G_{\oplus}$ due to the satisfied invariants stated at the beginning of this section. Now, we prove that $\mathcal{C}$ is a minimal clique partition. Let $\mathcal{I} = \{\mathit{Deep}(C) : C \in \mathcal{C}\}$: we claim $\mathcal{I}$ is an independent set in $G_{\oplus}$. The tree $\mathcal{T}$ induces a tree structure on $\mathcal{C}$. Consider two arbitrary cells C_1 and C_2 in $\mathcal{C}$ with a lowest common ancestor $x \in V$. If $\mathit{Deep}(C_1) \oplus \mathit{Deep}(C_2)$ then, due to Lemma 3.1, both $\mathit{Deep}(C_1) \oplus x$ and $\mathit{Deep}(C_2) \oplus x$ would hold. With Lemma 3.2 $\mathit{Deep}(C_1)$ would be compatible with every vertex on the path from $\mathit{Deep}(C_1)$ to x; the same would hold for $\mathit{Deep}(C_2)$. Hence, arriving at x, the algorithm would identify two children of x as $\mathit{Root}(C_1)$ and $\mathit{Root}(C_2)$ and finally merge these cells. Thus, the deep vertices of the cells in $\mathcal{C}$ are incompatible, i.e., independent in $G_{\oplus}$. As an independent set provides a lower bound for clique partitions and $|\mathcal{C}| = |\mathcal{I}|$ holds, $\mathcal{C}$ is an optimal clique partition of $G_{\oplus}$.

During the tree traversal, each vertex considers all its children once. This involves testing $x \oplus \mathit{Deep}(\mathit{Cell}(v))$. If we precompute the weight $w(P_{sv})$ for each vertex v, this test runs in constant time. This bounds the runtime by $\mathcal{O}(n)$. □

Algorithm 12: ContractTree

Data: Rooted tree $\mathcal{T}$ with edge weights.
Result: Minimum-cardinality allowed contraction $\mathcal{C}$.

$\mathcal{C} \leftarrow$ the set with a singleton cell for each vertex of $\mathcal{T}$;
forall vertices $x \in \mathcal{T}$, in post-order **do**
  $S \leftarrow \{Deep(Cell(v)) \mid v \in Children(x)\}$;
  $S_\oplus \leftarrow \{v \in S \mid v \oplus x\}$;
  Merge $Cell(x)$ and all $Cell(v)$ for $v \in S_\oplus$;
end
return $\mathcal{C}$;

Recall that a graph is called *chordal* if and only if all cycles of length at least 4 have a chord, i.e., an edge that is not part of the cycle but which connects two of its vertices. Now we prove that the compatibility graph $G_\oplus$ is chordal. This immediately gives a polynomial-time algorithm for TreeSummary, since CliqueCover can be solved in polynomial time on chordal graphs via its relation to coloring [Maf03]. We already have an easily-implementable linear-time algorithm based on the extra structure available to us (Algorithm 12). Still, the chordality of $G_\oplus$ is interesting from a structural point of view.

Theorem 3.2. *The compatibility graph $G_\oplus$ is chordal.*

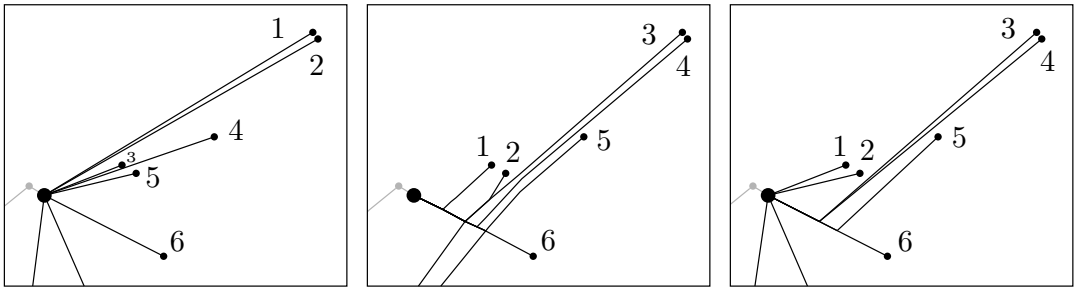
Proof. Consider an arbitrary cycle in $G_\oplus$ with vertex set $V_C \subseteq V$ and $|V_C| \geq 4$. Let $u, v, w \in V_C$ be a sequence of vertices on this cycle where $\{u, v\}, \{v, w\} \in E_\oplus$ are edges of the cycle and v is the deepest vertex, that is, $w(P_{sv}) \geq w(P_{su})$ and $w(P_{sv}) \geq w(P_{sw})$. Furthermore let $x_u \in V$ be the lowest common ancestor of u and v (considering $\mathcal{T}$), and let $x_w \in V$ be the lowest common ancestor of w and v. Without loss of generality we assume $w(P_{sx_w}) \geq w(P_{sx_u})$.

In this situation, with both x_u and x_w being ancestors of v, either $x_u = x_w$ or x_w is a descendant of x_u. First assume $x_u = x_w$. With Lemma 3.1 we derive from $v \oplus u$ and $v \oplus w$ that both v and w are compatible with x_u. Then, another application of Lemma 3.1 tells us that $u \oplus w$ and thus $\{u, w\} \in E_\oplus$. Now assume that x_w is a descendant of x_u. Since $u \oplus v$ and Lemma 3.1 hold, u and v are compatible to their lowest common ancestor x_u. With w being a descendant of x_u and $w(P_{sv}) \geq w(P_{sw})$, the compatibility $v \oplus x_u$ results in $w \oplus x_u$ because of Lemma 3.2. Again, we have v and w being compatible with x_u. With Lemma 3.1 this results in $u \oplus w$ and thus $\{u, w\} \in E_\oplus$.

We see in either case that V_C has a chord. Consequently, $G_\oplus$ is chordal. □

3.4 Map generalization

Now we return to map generalization and our example problem of finding a way to Groningen. Here, the TreeSummary problem does not directly apply: the road network we are interested in is unlikely to be a tree. Instead, we have a geometric graph $G = (V, E)$ with



(a) Direct. (b) Detailed. (c) Simplified.

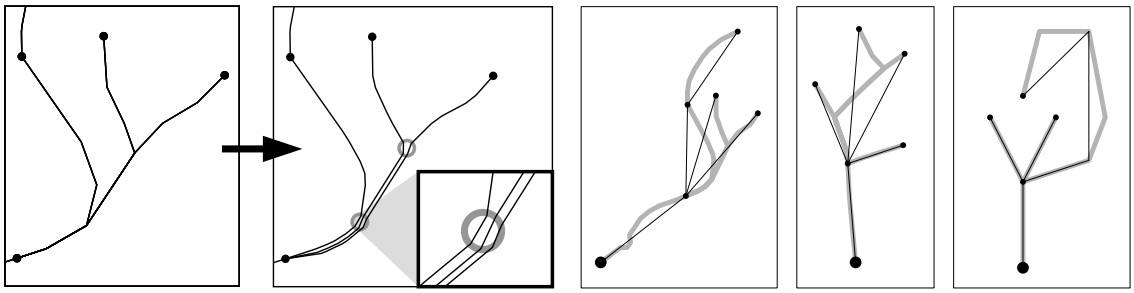
Figure 3.3: Using direct line segments between cell roots is not topologically safe. Even if there are no intersections in the output, it may still be the case that the embedding changes as illustrated in this example: the children of the fat vertex are numbered in clockwise order.

the user's position $s \in V$ and a nonnegative weight function $w\colon E \to \mathbb{R}_{\geq 0}$. (In order to generate destination maps [KAB+10] instead, we can pick s to be the destination.) In this section, we will first consider a tree in G, so that we may apply our tree contraction algorithm. Afterward we reintroduce the connectivity information from G in order to generate the generalized map.

Since the generalized map is intended for planning routes, a shortest-path tree of G (rooted at s) seems appropriate. However, if we simply calculate a normal shortest-path tree, we lose a lot of information. Consider an edge $e = \{u, v\} \in E$, where neither the shortest path to u passes through v nor the other way around. This edge would not be included in a normal shortest-path tree. This is correct if we only consider the vertices of the graph. However, if the graph represents a continuous road network, then there is in fact a point on the interior of the edge at which s is equidistant through u and through v; anywhere else on the edge, one path or the other is shorter.

In order to incorporate this connectivity information in our visualization of the road network, we start by calculating a modified shortest-path tree $\mathcal{T}' = (V', E')$ where $V \subseteq V'$. This tree $\mathcal{T}'$ contains all edges from the normal shortest-path tree, but as noted, there are edges in E that do not show up. For every such "missing" edge $e = \{u, v\}$ we do the following: we find the position on e where the paths to s via u and via v have the same length, where we assume the weight of e is distributed uniformly along its length. (This holds, of course, if the weights are Euclidean distances. If, for another example, the weights represent travel time, the assumption is still reasonable.) At this position on e, we introduce two *virtual vertices* p_1 and p_2 to V' and add the edges $\{u, p_1\}$ and $\{p_2, v\}$ to E', assigning weights appropriately. (To contrast these virtual vertices, the original vertices of V are called *real*.) By annotating p_1 and p_2 with references to the other, the resulting tree $\mathcal{T}'$ represents all edges and connectivity of G.

After constructing $\mathcal{T}'$, we apply Algorithm 12 to solve TreeSummary on it. The result is a partition of V' into contractible cells. These cells are highly detailed close to s and less detailed the farther away they are. The question remains how to draw a generalized map based on these cells.



(a) Peeling a tree.

(b) Some simplifications. Fat gray lines indicate the input; black lines indicate the output.

Figure 3.4: Simplifying a tree as a set of polylines. Dots indicate cell roots.

We represent each cell by its root vertex, which we draw in its original position given by G. Note that the cells themselves are also related in a tree structure induced by $\mathcal{T}'$: call a cell C the child of another cell P if and only if $Root(P)$ is the first cell root encountered on the path from $Root(C)$ up to the tree root s. We propose the following three visualizations of $\mathcal{T}'$.

- For every cell C and its parent cell P, let $c = Root(C)$, $p = Root(P)$ and draw a straight line segment between c and p. We call this the *direct* drawing. The advantage of this drawing is that it is simple, highly generalized, and has a one-to-one correspondence between cells and output vertices. We do note that in general these line segments may intersect and that, even if they do not, the embedding of the output map may be inconsistent with the input. (See Figures 3.3 and 3.4b for an illustration.) This poses problems for a road map: intersecting line segment may visually imply connectivity, and even if the appropriate details will show up once the user comes closer, an incorrect embedding may negatively impact the user's mental model.
- For every cell C and its parent cell P, let $c = Root(C)$, $p = Root(P)$ and draw P_{cp}. That is, draw the path in $\mathcal{T}'$ that connects c and p. We call this the *detailed* drawing. Note that P_{cp} is an actual shortest path in G. This is a positive aspect of the detailed drawing: it is in fact an edge selection of G and it contains those parts of the shortest path tree that connect the cell roots. This does mean that, even though a selection has been made, the selected polylines still have the same level of detail as the input, which may or may not be desirable.

  It should also be noted that this drawing can have vertices of degree larger than two that are not at a cell root: we call these *internal branches*. (See Figures 3.3 and 3.4 for an illustration.) The existence of such vertices can be considered somewhat unfortunate after we have carefully optimized the selection of cell roots, but this drawing is true to G and it does not necessarily look displeasing.
- Construct the detailed drawing and apply a topologically-safe simplification algorithm. We call this the *simplified* drawing. We use the algorithm of Dyken et al. [DDS09], which greedily deletes vertices but never moves them, and never deletes cell roots. We run the algorithm without stopping criterion, that is, as far as it will go. In order to



(a) Input. (b) Detailed. (c) Simplified.

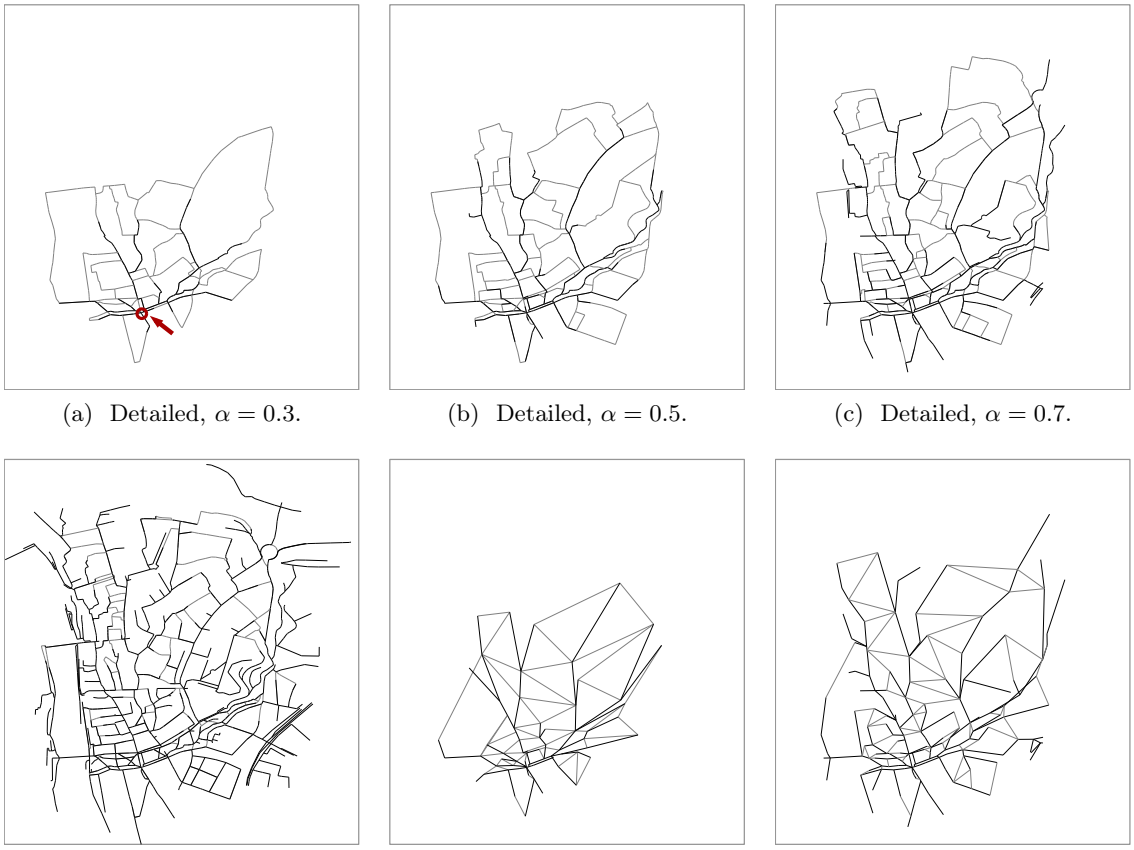
Figure 3.5: Würzburg with $\alpha = 0.8$, with the vertex s indicated in figure (a). The simplified drawing contains only two internal branches: Figure 3.3 is a crop of this map showing the extra vertices (located near the bottom right of this figure).

(hopefully) get rid of any internal branches that the detailed drawing might have, we slightly offset each path so that they do not actually touch except in cell roots and therefore do not actually form internal branches. (Topologically, this can always be done and it can be implemented simply in practice if one accepts some inefficiencies. It has not been our goal to optimize this step.) We call this the peeled tree: see Figure 3.4(a). After simplification we merge any remaining vertices that logically belong to the same vertex in V'. This may introduce internal branches, but ensures that the final drawing does not contain infinitesimally-shifted paths

Figure 3.5 shows a map of the German city of Würzburg, and *detailed* and *simplified* drawings for the same root s. It can be seen that the detailed drawing already provides a focus-and-context effect, giving more detail near the bottom of the map, slightly left of the middle, which is where s is located. When compared to the simplified drawing, it can be seen that internal branches are present. Simplifying those gives a much cleaner output graph, but naturally this comes at some cost to the recognizability of the road network. In this particular example, only two internal branches remain in the simplified drawing.

When drawing maps, restricting ourselves to outputting trees limits the information we can present. Indeed, as we argued at the start of the section, we should care about the cross links between the cells: between cells that share a virtual vertex, to be precise, since these are cells that touch in the network G. This occurs on the interior of edges in E (or in vertices of V) and it is actually possible to navigate from a cell to a touching cell. We now draw these adjacencies, either *directly*, with a straight-line segment, or with a *detailed* path: from one cell root, to the virtual vertex, and then to the other cell root. If a pair of cells touch in more than one virtual vertex, we draw this connection only for a single virtual vertex that has the shortest distance to s.

Figure 3.6 shows the resulting drawing of Würzburg, for varying values of α. More so than the tree drawing of Figure 3.5, this resembles a traditionally useful map and it also exhibits a focus-and-context effect, the strength of which depends on α.



(a) Detailed, $\alpha = 0.3$. (b) Detailed, $\alpha = 0.5$. (c) Detailed, $\alpha = 0.7$.

(d) Detailed, $\alpha = 0.9$. (e) Direct, $\alpha = 0.5$. (f) Direct, $\alpha = 0.7$.

Figure 3.6: Würzburg with cross connections, for various values of α, and with either detailed of direct drawing style. All drawings use the same vertex s, which is indicated in figure (a). Note that figures (b) and (e) are the detailed and the direct drawing of the same underlying partition; this also holds for (c) and (f).

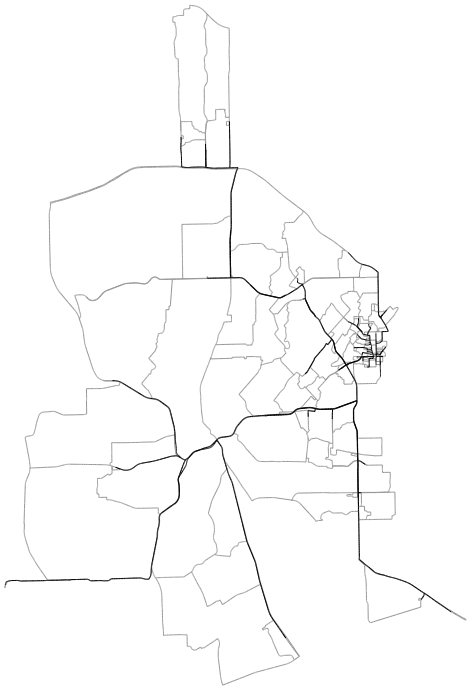
Figure 3.7(a) shows a map of a much larger road network (Dallas, Texas). Figure 3.7(b) shows a *detailed* drawing, with cross connections and $\alpha = 0.5$. This already gives a focus-and-context effect by having full detail in an area of interest and a gradually more generalized road selection as the (network) distance increases. Figure 3.7(c) shows an example of how this effect can be enhanced and makes the map more useful by computing a variable-scale map transformation in the style of Haunert [HS11, vDvGH$^+$13].

We finish with an observation about the abovementioned cross connections. In order to provide meaningful information to the user, we want to give an estimation of how long a detour might get if the user chooses to use the shortest path to a neighboring cell and such a cross link instead of following the route on $\mathcal{T}'$.

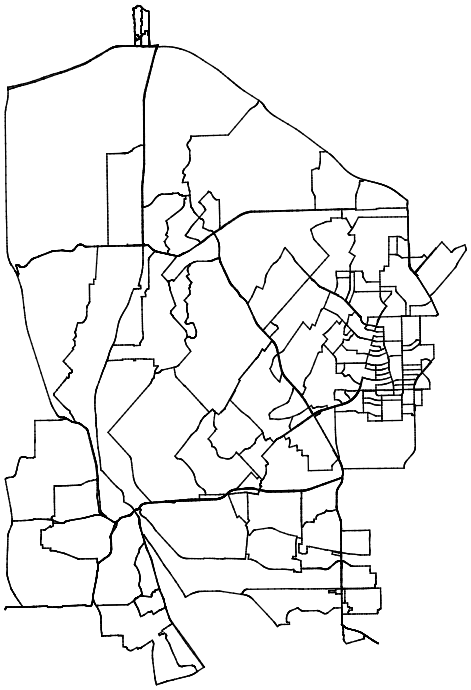
Theorem 3.3. *A detour through a neighboring cell is at most* $\left(\frac{2}{\alpha} - 1\right)$ *times as long as the shortest path.*



(a) Input network.



(b) Detailed drawing, $\alpha = 0.5$.



(c) Focus map.

Figure 3.7: The city of Dallas, Texas. In Figure (b), the black lines indicate a detailed drawing of the tree summary and the gray lines indicate cross connections. The focus map (c) was computed with the method of van Dijk et al. [vDvGH+13], using a scale factor of 3 for a region around the selected center.

Proof. Consider an arbitrary vertex $u \in V'$, either real or virtual. Let $v \in V' \setminus V$ be any virtual vertex in the same cell as u (possibly $u = v$) and let $r \in V$ be the root of the cell containing u. Then the detour of going through v is a factor $\delta := w(P_{uvs})/w(P_{us})$. We wish to bound δ.

We first observe by the triangle inequality for shortest paths that the distance from u to v is at most the distance from u to v via r. This detour is possible since u and v are both in the cell with root r. This gives

$$1 \leq \delta \leq \frac{w(P_{urvs})}{w(P_{us})} = \frac{w(P_{ur}) + w(P_{rv}) + w(P_{vs})}{w(P_{ur}) + w(P_{rs})}.$$

Since all the weights are nonnegative and u influences the right-hand side only through $w(P_{ur})$, we can maximize the given ratio by setting $w(P_{ur}) = 0$, since $w(P_{ur})$ appears in both the numerator and the denominator as a summand. (That is, the detour is worst when u is a cell root, which makes sense.)

We observe that $w(P_{rv}) = w(P_{vs}) - w(P_{rs})$ because r lies on P_{vs}. Since v and its ancestor r lie in the same cell, they are compatible and we have $w(P_{vs}) \leq \frac{1}{\alpha} w(P_{rs})$. Putting these things together,

$$\begin{aligned}\delta &\leq \frac{w(P_{rv}) + w(P_{vs})}{w(P_{rs})} = \frac{\big(w(P_{vs}) - w(P_{rs})\big) + w(P_{vs})}{w(P_{rs})} \\ &\leq \frac{\big(\big(\frac{1}{\alpha} - 1\big) + \frac{1}{\alpha}\big) \cdot w(P_{rs})}{w(P_{rs})} = \frac{2}{\alpha} - 1.\end{aligned}$$

This is the bound in the theorem. □

For $\alpha = 0.95$, as proposed in the introduction, the above theorem gives an upperbound of 11% on any detour, which seems very reasonable. The value $\alpha = 0.9$ results in 22% and even for $\alpha = 0.5$ we still get a factor 3.

Experimental setup

We have implemented Algorithm 12 in *C++*. To generate the simplified drawings, we have used CGAL's implementation [Fab15] of a topologically-safe simplification algorithm by Dyken et al. [DDS09]. The computations for this paper were run on a desktop PC with an Intel® Core™ i5-2400 CPU at 3.10 GHz; memory usage was not an issue. The map of Würzburg is a crop of the *OpenStreetMap* road network of Würzburg, Germany (`http://download.geofabrik.de`). It uses Euclidean weights and contains approximately 2500 vertices. The runtime of both Algorithm 12 and the simplification is instant. The map of Dallas is the largest connected component in the *City of Dallas GIS Services*' road network of Dallas, Texas (`http://gis.dallascityhall.com/EnterpriseGIS`). It has approximately 3×10^5 vertices and the weights are a travel-time estimate based on road class. Algorithm 12 still runs almost instantly, given $\mathcal{T}'$. The entire computation, which includes building $\mathcal{T}'$ using an unoptimized implementation and the simplification, has a runtime of about 2 seconds. We conclude that this approach is suitable for interactive applications on realistically-sized maps.

3.5 Conclusion and Outlook

We have proposed partitioning the vertices of a tree based on whether or not they share a significant part of their path to the root. We gave a linear-time algorithm for minimizing the number of cells in this partition and proved several properties of such clusterings. This approach could be used to model various clustering problems on trees, such as summarizing large hierarchies (organizational charts, file systems, et cetera). We looked specifically at a map generalization problem where we summarize a shortest-path tree and were able to derive a constant-factor bound on the "generalization error" as measured by the detours resulting from using the output map.

We would argue that our visualization of the computed clustering looks promising, but is certainly open for improvement. A more elegant solution to the topological issue of internal branches would be welcome. After all, we have taken care to compute an optimal solution to the TreeSummary problem: we have already decided what we *want* to draw. Given a desire for topological safety and without moving any vertices, we may be forced to include additional detail. Letting go of exact geographic positions may lead to better results by allowing map deformations. What deformations are appropriate, and how to compute them, would clearly depend on the application. Another direction would be a visualization where the clusters are represented by areal features.

Other map generalization scenarios present themselves depending on which information is available to us. Say we are not only given the weighted graph $G = (V, E)$, but additionally get a partition $\mathcal{P} = \{V_1, \ldots, V_k\}$ of the vertex set, where each cell of $\mathcal{P}$ reflects a geographic region (such as "Groningen"). The task is then to decide for each cell $V_i \in \mathcal{P}$ whether or not the vertices in V_i are equivalent, for which we might use our contractibility condition. Depending on the result, V_i will be collapsed and displayed as a single vertex or each vertex in V_i will be displayed individually. One can also consider the case where a hierarchical partition is given (neighborhoods, cities, states, federations). An advantage of this setting is that the cells are likely to have meaningful names, which will help when labeling the map. This raises the open question of how labels for geographic regions can be carried over to a clustering computed using our algorithm. As a disadvantage of the pre-clustered approach, it is more constrained in its options than our approach, so it will use at least as many clusters and probably more.

Another direction for future work is to consider a dynamic version of the problem, where the vertex s changes. This is relevant, for example, when using this visualization in an in-car navigation system. Our algorithms are fast enough to be run in realtime on realistic maps, but this does not close the case: for interactive or animated visualizations, it is important to consider the stability of the computed solutions. This should be considered as an optimization problem, but is currently open.

4 A Cutting-Plane Method for Contiguity-Constrained Spatial Aggregation

The following chapter is mainly taken from a joint work with Jan-Henrik Haunert [OH17]. In this article, we deal with an **NP**-complete problem of aggregating areas. For the aggregation process, we consider both semantic and geometric, in particular topological, aspects.

Due to its complexity, solving this problem exactly is time-consuming. Consequently, various heuristic approaches exist [HWS+65, MCVL02, BEL03, DAR12, LCG14]. Attempts to solve this problem (or similar ones) exactly can be found in the existing literature and are commonly based on integer linear programming [CCG+13, DH99, DCM11, Shi05a]. We present a new exact approach which is also based on integer linear programming. For this purpose, we apply cutting-plane techniques which are established in the field of combinatorial optimization to the considered spatial aggregation problem.

Abstract

Aggregating areas into larger regions is a common problem in spatial planning, geographic information science, and cartography. The aim can be to group administrative areal units into electoral districts or sales territories, in which case the problem is known as districting, but often area aggregation is seen as a generalization or visualization task, which aims to reveal spatial patterns in geographic data. Despite these different motivations, the heart of the problem is the same: Given a planar partition, one wants to aggregate several elements of this partition to regions. These often must have or exceed a particular size, be homogeneous with respect to some attribute, contiguous, and geometrically compact. Even simple problem variants are known to be NP-hard, meaning that there is no reasonable hope for an efficient exact algorithm. Nevertheless, the problem has been attacked with heuristic and exact methods.

In this article we present a new exact method for area aggregation and compare it with a state-of-the-art method for the same problem. Our method results in a substantial decrease of the running time and, in particular, allowed us to solve certain instances that the existing method could not solve within five days. Both our new method and the existing method use integer linear programming, which allows existing problem solvers to be applied. Other than the existing method, however, our method employs a cutting-plane method, which is an advanced constraint-handling approach. We discuss this approach in detail and present its application to the aggregation of areas in choropleth maps.

4.1 Introduction

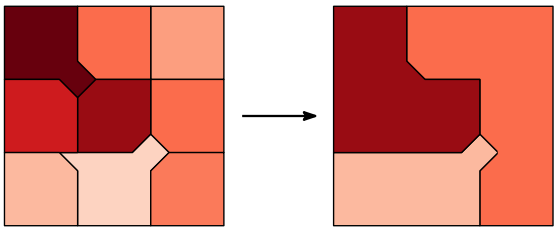
Planar subdivisions are frequently used to structure geographic space. In geographic information systems, they can be used as a basis for data acquisition, storage, analysis, and visualization. Since different applications require information on different scales, planar subdivisions are often hierarchical—unemployment rates, for example, can be analyzed on a county or country level. Often, one aims to compute a higher-level subdivision from a given one, by grouping areas of similar attribute values. With such an approach, one can reveal large-scale patterns in the data. In this article, we present a new method for area aggregation, which we discuss in the context of districting and spatial unit allocation. We distinguish these problems as follows:

- *Districting* [HS71, CSGW04, Hoj96, Shi09] is the problem of partitioning a set of minimum mapping units (e.g., postal code zones) to form larger regions or districts (e.g., school zones or electoral districts). The minimum mapping units (i.e., input areas) are assumed to form a planar subdivision. The applications of districting range from administrative to commercial purposes; an overview is provided by Shirabe [Shi05a].
- *Spatial unit allocation* [Shi05b, Shi05a] subsumes districting, but it does not necessarily ask to assign every area to a district. A typical example of spatial unit allocation is to select a set of areas constituting a single region that is geometrically compact and requires minimal development costs [AEHS03].
- The term *area aggregation* has been used to refer to the aggregation of areas as a data abstraction or map generalization problem [HW10a]. Just as districting, area aggregation requires a planar subdivision as input and asks to group the areas into larger regions. Districting problems in spatial planning, however, do not necessarily ask to group areas of similar attribute values, which is an essential criterion for generalization.

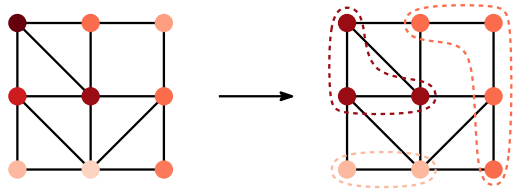
It is common to approach districting, spatial unit allocation, and area aggregation by optimization [MJN98, BEL03, GN70, Hau08, Shi09, DCM11, LZCJ08]. The existing approaches are quite similar, since the different problem variants often share some optimization objectives and constraints. For example, it is common to require that every output region must have a size or population within certain bounds and to favor geometrically compact shapes. Compactness is usually assessed quantitatively, which can be done with different measures [Mac85, LGC13], and considered as an optimization objective. Additionally, in many problem variants, the output regions are required to be *contiguous* [Shi09, Shi05a, Wil02, DCM11, CC10]:

Definition 4.1. *An area $A \subseteq \mathbb{R}^2$ is called* contiguous *if every two points in A are connected via a (not necessarily straight) line that is contained in A.*

Though the output regions tend to become contiguous when compactness is considered as an objective, there is generally no guarantee for contiguity without enforcing it. In fact, if compactness is not the primary objective, contiguity often has to be enforced to produce somehow reasonable output regions [Shi09]. Therefore, we think that optimizing similarity



(a) Explanatory maps: input map (left) and output map (right).



(b) The adjacency graph $G = (V, E)$ for the input map in (a) and the partition of V corresponding to the output map.

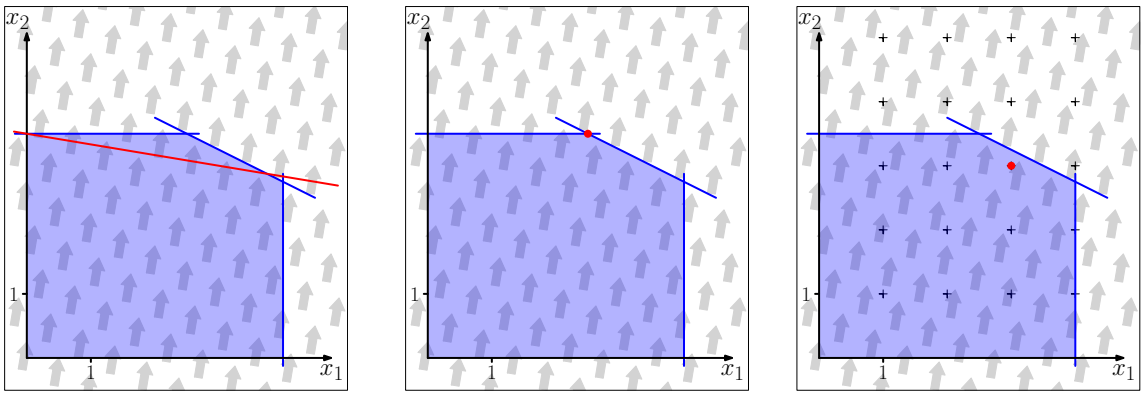
Figure 4.1: An example of area aggregation (see Haunert and Wolff [HW10a]).

(and compactness as a secondary criterion) subject to size constraints and contiguity is a particularly interesting challenge.

Attribute similarity is an important criterion for aggregation in *map generalization*, which means to generate a more abstract and less detailed representation of geographic space from a given one [BW88, Wei97, Ses05]. The problem occurs if the scale of a cartographic visualization has to be reduced, but map generalization does not necessarily assume that the input and output representations are visual graphics. Haunert and Wolff [HW10a] have defined the area aggregation problem in map generalization formally and developed an optimization method for it, which is based on models for districting by Zoltners and Sinha [ZS83] and Shirabe [Shi09]. The problem not only requires to group the input areas into larger regions, but also to assign a value from the attribute domain to each output region; see Fig. 4.1(a). The aim is to minimize a cost function that penalizes changes of attribute values to dissimilar values as well as geometrically non-compact output regions, subject to constraints concerning the size and contiguity of the output regions. The method relies on the definition of the adjacency graph $G = (V, E)$ whose vertex set V contains a vertex for each input area and whose edge set E contains an edge $\{u, v\}$ for each two adjacent areas $u, v \in V$; see Fig. 4.1(b). We will use this definition of G throughout this article.

In this article, we revisit the problem defined by Haunert and Wolff [HW10a], but we also consider the special case that similarity is neglected and compactness is the sole objective. In this case, the problem is more similar to a classical districting problem that simply demands geometrically compact and contiguous regions of sizes within certain bounds. Moreover, while Haunert and Wolff developed and tested their method for the generalization of categorical maps, we will use our method to generalize choropleth maps with a ratio-scaled variable, such as unemployment rates. This has the advantage that we can directly compute differences between attribute values and do not depend on the definition of a semantic distance between categories.

We focus on *exact* optimization methods based on *integer linear programming*, which is a common optimization approach for districting [CSGW04, HS71, Hau08]. In particular, it is reasonable for NP-hard problems, for which the existence of an efficient and exact algorithm is extremely unlikely [GJ90]. In fact, area aggregation falls into the class of NP-hard problems [HW10a], and so do many problem variants of districting [Alt97, PT08, Hoj96].



(a) The red line marks (x_1, x_2)-values with equal objective value.

(b) A solution to the linear program.

(c) A solution to the integer linear program. The LP in (b) is completed with $x_1, x_2 \in \mathbb{Z}$.

Figure 4.2: A linear program with variables x_1 and x_2; objective is to maximize $6x_1 + x_2$ with respect to $x_1 \geq 0$, $x_2 \geq 0$, $2x_2 \leq 7$, $2x_1 + 4x_2 \leq 18$ and $x_1 \leq 4$. The solution space, i.e., all pairs (x_1, x_2) for which every given inequality is true, is marked as a blue polytope. The gray arrows in the background indicate the objective. They are orthogonal to the line of equal objective values in (a).

Defining an *integer linear program* (ILP) means setting up a *linear program* (LP) completed with an *integrality constraint*. While the LP defines a linear objective function and a set of linear inequality constraints over a set of variables, the integrality constraint requires that the variables receive integer values. An ILP is solved optimally if a variable assignment is found which optimizes the objective function without violating any constraint. Commonly, the LP corresponding to an ILP without integrality constraints is referred to as the *LP relaxation* of the ILP [NW88]. An illustrative example for both an LP and an ILP can be found in Fig. 4.2.

Whereas efficient algorithms for linear programming exist, integer linear programming is NP-hard [CLR90, GJ90]. Nevertheless, an approach based on integer linear programming is promising as it allows sophisticated optimization software (e.g., CPLEX [CPL17] and Gurobi [Gur18]) to be applied. Even though the exact methods for solving ILPs have an exponential worst-case running time, they can be relatively fast when applied to real-world instances. Moreover, solutions of an exact method can be used as quality benchmarks to evaluate the results of (faster) heuristic algorithms. Heuristic methods for districting have been developed by several researchers [HWS+65, MCVL02, BEL03, DAR12, LCG14]. In contrast to exact methods, these do not guarantee to deliver an optimal solution.

Just as some criteria are shared by many problem variants of districting and spatial unit allocation, the ILP formulations for these problems often share some elementary components. Shirabe [ST02] has used this fact to integrate mathematical programming techniques and geographic information systems (GIS), such that a GIS user can assemble a model for a particular spatial unit allocation problem from a set of elementary model components and compute a solution to the problem with an ILP solver. Since contiguity is an impor-

tant requirement in many spatial unit allocation problems, several works have focused on formalizing contiguity as one such elementary model component [Shi09, Shi05a, Wil02].

Usually, there exist multiple possibilities of encoding a particular problem as an ILP; choosing among these ILP formulations can highly influence the computation time. In geographic information (GI) science, it is common to choose a *compact* ILP formulation, which means that the size of the ILP is polynomial in the size of the input [FM12]. For example, in Section 4.2.2 we show that, when applying Shirabe's model [Shi05a] to area aggregation without prescribing the number of output regions, it has $\mathcal{O}(n^2)$ variables and constraints (where n is the number of input areas) and thus a polynomial size. A compact ILP formulation can be favorable because it permits a full instantiation of the model, that is, a file or data structure explicitly storing all variables and constraints. After the instantiation of the model, it can be handed over to a solver, which computes an optimal solution without requiring any further interaction. It is therefore common to think of the solver as a black box [LR05].

Working with compact ILP formulations is relatively convenient. It is also known, however, that they are sometimes outperformed by *non-compact* ILP formulations, whose number of constraints can be exponential in the size of the input [Pat03]. Such a large set of constraints forbids a full instantiation of the model. Therefore, one starts the computation by working with a reduced ILP, which is lacking a set of constraints of the original ILP. A simple approach is to solve this reduced ILP to optimality and to examine whether the solution violates constraints of the original ILP. If a violation of a constraint is found, that constraint is added to the ILP and the solution process is started anew. Drexl and Haase [DH99] and Duque et al. [DCM11] use this approach to solve districting problems. In particular, Duque et al. deal with the *p-regions problem*, in which, other than in our problem, the number p of output regions is prescribed.

In this article, we present a more sophisticated approach for area aggregation. We demonstrate the effectiveness of a *cutting-plane method*, which generally refers to a method that adds constraints during optimization without relying on an optimal solution to the reduced ILP. Violated constraints are found already in a preliminary stage of a solution, namely in an optimal solution to the LP relaxation of the reduced ILP. Such constraints are termed *cutting planes* (or simply *cuts*) because they cut away parts of the feasible region of the solution space defined by the current instantiation of the model. The number of constraints of our ILP formulation for area aggregation is exponential in the number n of input areas, but initially we instantiate the model with only $\mathcal{O}(n^2)$ constraints. We generate constraints ensuring contiguity during optimization, using what is generally termed a *separation algorithm* [NW88]. Though implementing a cutting-plane approach requires some understanding of how an ILP solver works and certainly more effort than using a compact ILP formulation and a black-box solver, we consider it practicable, also for researchers in GI science. This is because modern ILP solvers such as CPLEX or Gurobi offer programming libraries that include interfaces (usually termed *callbacks*) for intervening in the optimization process. Carvajal et al. [CCG+13] have developed a cutting-plane method based on an efficient separation algorithm for a problem of spatial unit allocation in forest management. We are not aware of such a method for area aggregation or districting, though.

We used the cutting plane method for contiguity-constrained spatial unit allocation by Carvajal et al. as a starting point, which, compared to the method of Drexl and Haase [DH99], is more recent and can be considered more sophisticated. However, we had to extend the method substantially for the case of an unknown number of output regions and a flexible set of centers. More precisely, Carvajal et al. consider two constraint formulations for the contiguity of a region, namely one that does not rely on the concept of a center of a region and one that requires a prescribed center to belong to the output region. While in the first model, contiguity is ensured by considering a set of constraints for each pair of nodes that are selected for the output region, in the second model, a set of constraints for each selected node ensures its connectivity to the prescribed center. Conceptionally, our approach is more similar to the second model, as it also relies on the idea of centers. However, we do not know in advance which of the nodes become centers. Therefore, we use a constraint formulation that, in fact, is more similar to the first model of Carvajal et al. in the sense that it uses one set of constraints for each pair of nodes.

Álvarez-Miranda et al. [ÁMLM13] and, recently, Wang et al. [WBB17] have presented theoretical findings on integer programming formulations for the problem of selecting a maximum-weight connected subgraph of a given graph. Though their results cannot easily be transferred to other problems, they can be understood as hints on why a non-compact ILP formulation, such as the one of Carvajal et al. or ours, can outperform a compact ILP formulation.

To summarize our contribution, we discuss cutting-plane methods as a general constraint-handling technique that is rather unknown in geographic information science but well established in the field of combinatorial optimization [vRW87, JRT95]. We show that our cutting-plane method outperforms the districting method of Shirabe [Shi09] that was adapted by Haunert and Wolff [HW10a] for the aggregation of areas in map generalization. For example, with our method we were able to solve various instances with 94 departments of France (excluding overseas department and the island of Corsica) in reasonable time, whereas using Shirabe's method produces a result in much longer time or not at all (see Section 4.5). We specifically apply this to generate a map that shows a structuring of France into a few (e.g. 10) regions of similar unemployment rates and thereby highlight the usefulness of the method for the generalization of choropleth maps. We do note that the applicability of our method is limited, since we were not able to process instances larger than our instances of France. The number of areas in these instances of France, however, can be considered typical for choropleth maps. Similar maps can be found, for example, in Bertin's fundamental textbook on visualization [Ber83].

In the following, we review an existing ILP formulation for area aggregation (Section 4.2). Then, we give an overview of strategies for handling ILPs with large sets of constraints (Section 4.3). Subsequently (Section 4.4), we contribute an ILP applying cutting planes which extends the ILP formulation from Section 4.2. Afterwards, we let both models compete in a series of experiments (Section 4.5). We apply both ILP formulations on a real-world example with 94 input areas, discuss the solutions, and compare the running times for different settings. We finish this article with concluding remarks and ideas for further improvements (Section 4.6).

4.2 A state-of-the-art model

The huge amount of work on spatial unit allocation and districting disallows a comprehensive review in this article. Therefore, we refer to the survey by Ricca et al. [RSS13] for an overview and discuss only the most relevant related work that has inspired our models. This in particular concerns a general districting model with assignments of areas to centers (Section 4.2.1) and a flow-based model to ensure contiguous output regions (Section 4.2.2). In Section 4.2.3, we briefly review the approach of Haunert and Wolff [HW10a] for the aggregation of areas in map generalization, which is based on the models from Sections 4.2.1 and 4.2.2.

4.2.1 A compact ILP without contiguity

The ILPs that we use in this article differ only with respect to the constraints ensuring contiguity. If we drop those constraints, we obtain an ILP that has the same structure as the *basic ILP* defined by Haunert [Hau08]. This basic ILP follows the approach of Zoltners [ZS83] for districting, in the sense that in every output region one of the input areas is selected as the region's center. To encode this idea, the basic ILP uses a variable $x_{c,v}$ for each pair of areas $c, v \in V$, which has the following meaning.

$$x_{c,v} = \begin{cases} 1, & \text{if area } v \text{ is assigned to the output region with center } c, \\ 0, & \text{otherwise.} \end{cases}$$

For $c = v$, the variable $x_{c,c}$ expresses whether area c is assigned to itself, meaning whether or not it is selected as a center. Note that with this model we do *not* prescribe the centers before computing an optimal assignment. Instead, every area can become a center.

Each variable $x_{c,v}$ is associated with an assignment cost $a_{c,v}$. The objective function is a weighted sum of the variables.

$$\min \sum_{c \in V} \sum_{v \in V \setminus \{c\}} a_{c,v} \cdot x_{c,v} \tag{4.1}$$

The aim for compact output regions, for example, can be expressed by minimizing this objective function with $a_{c,v} = w(v) \cdot d(c, v)$, where $w(v)$ is a weight for area v (reflecting size or population) and $d(c, v)$ is the Euclidean distance between the centroids of c and v [HW10a].

To obtain a partition of the set V of areas into regions, we require with the following constraint that every area is assigned to exactly one center.

$$\sum_{c \in V} x_{c,v} = 1 \quad \text{for each} \quad v \in V \tag{4.2}$$

Next, we make sure that an area v is assigned to a center $c \in V$ only if c is actually selected as a center.

$$x_{c,v} \leq x_{c,c} \quad \text{for each} \quad c \in V,\ v \in V \setminus \{c\} \tag{4.3}$$

To impose constraints on the size or population of each output region, we use the weight $w(v)$ that we defined for Objective (4.1). The following constraint ensures that every output region has a weight of at least $w_{\min} \in \mathbb{R}^+$.

$$\sum_{v \in V} w(v) \cdot x_{c,v} \geq w_{\min} \cdot x_{c,c} \quad \text{for each} \quad c \in V \tag{4.4}$$

Similarly, we could define an *upper* bound on the weights of the regions.

Interpreting both the previous constraints and the results makes sense only if the integrality constraint (Constr. (4.5)) is taken into account.

$$x_{c,v} \in \{0,1\} \quad \text{for each} \quad c \in V,\ v \in V \tag{4.5}$$

A solution satisfying this constraint is termed an *integral solution*.

4.2.2 Shirabe's model for contiguity-constrained spatial unit allocation

The model presented in this section is a straightforward adaption of a model for spatial unit allocation by Shirabe [Shi05a, Shi09]. It extends the basic ILP from Section 4.2.1 with a set of continuous variables and an additional set of constraints to ensure contiguous regions. Since the additional variables are continuous, Shirabe's model leads to a *mixed* integer linear program (MILP), which basically can be solved with the same solution techniques as an ILP. In contrast to Shirabe, we neither demand a single output region [Shi05a] nor a partition of the input graph into a prescribed number of regions [Shi09] and, thus, have to make small modifications. We denote the resulting MILP as the *flow MILP* and will use it as a benchmark to evaluate our new cutting-plane method.

The flow MILP relies on the definition of the directed graph $\bar{G} = (V, \bar{E})$, whose set $\bar{E}$ of directed edges (or *arcs*) contains arc (u, v) as well as arc (v, u) for every edge $\{u, v\} \in E$. It uses the idea that multiple commodities are transported (or *flow*) on the arcs of this graph. By controlling the flow of the commodities with suitable constraints, the contiguity of the output regions is ensured.

In our adaption of Shirabe's model, there is one commodity for each potential region center—and thus for each vertex $c \in V$. We define a variable

$$y^c_{(u,v)} \in \Big[0, |V| - 1\Big] \quad \text{for each } (u,v) \in \bar{E},\ c \in V \setminus \{u\},$$

which represents the amount of the commodity for center c that flows on arc (u, v). Every area v that is assigned to a region center $c \neq v$ is a *source*, that is, it injects one unit of the commodity for c into the flow network. This unit flow is forced to find a way to the region center c—the sole *sink* for the commodity for c—by only passing through areas allocated to the same center. Guaranteeing these properties of the flow is equivalent to guaranteeing the contiguity of the resulting regions. This is done with the following constraints:

$$\sum_{(u,v) \in \bar{E}} y^c_{(u,v)} \leq \big(|V| - 1\big) \cdot x_{c,u} \qquad \text{for each} \quad c \in V,\ u \in V \setminus \{c\} \tag{4.6}$$

$$\sum_{(u,v) \in \bar{E}} y^c_{(u,v)} - \sum_{(v,u) \in \bar{E}} y^c_{(v,u)} = x_{c,u} \qquad \text{for each} \quad c \in V,\ u \in V \setminus \{c\} \tag{4.7}$$

If $x_{c,u} = 0$, i.e., u is not assigned to center c, Constraint (4.6) prohibits any outflow of the commodity for c from u; Constraint (4.7) forces the inflow and the outflow of this commodity at u to be equal (and thus prohibits any inflow as well). If $x_{c,u} = 1$, i.e., u is assigned to center c, Constraint (4.7) ensures that u is a source contributing one unit of the commodity for c to the flow network; Constraint (4.6) is relaxed by setting its right-hand side to a sufficiently large number. Only the center c of a region can be a sink of the commodity for c, since Constraints (4.6) and (4.7) are not set up for the case $c = u$.

The flow MILP consists of Objective (4.1) as well as Constraints (4.2)–(4.7). Since G is planar, the flow MILP has $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^2)$ constraints, where $n = |V|$ is the number of input areas.

4.2.3 Area aggregation in map generalization

Area aggregation is an important sub-problem of map generalization, which (among others) also involves line simplification [dBvKS98], selection [MB93], and displacement [Ses05, BBW05]. While some approaches exist to treat all or multiple sub-problems of map generalization in a comprehensive way [Gal03, WJT03, HW07], research is also ongoing to improve the algorithmic solutions for each sub-problem.

Area aggregation can be driven by minimal graphic dimensions for a target scale, but in this article we consider it in the context of *statistical generalization* [BW88], which aims at a less detailed and non-graphical model to permit spatial analysis on a higher level of abstraction. In particular, we consider the grouping of administrative regions with unemployment rates as an example, in which the aim is to reveal large spatial patterns of unemployment. Our aim is a high homogeneity with respect to the unemployment rate in each output region, thus we consider similarity of attributes as an important criterion for grouping. Additionally, we consider size and compactness in our model.

We do note that grouping areas based on unemployment rates is related to the delineation of *labor market areas*, which, however, are usually defined based on travel-to-work patterns rather than on attribute similarity [Sma74, CD00, FRCDMB08, PN02]. Our work is also related to the modifiable areal unit problem (MAUP) [Ope84], which states that the delineation of districts is a source of statistical bias. With our aim for homogeneous output regions we try to keep this bias low. Obviously, aggregation not only reveals large spatial patterns but also suppresses fine-grained and possibly sensitive information. Therefore, we also see a relevance of area aggregation for privacy protection [KCS04].

According to the definition of area aggregation by Haunert and Wolff [HW10a], the areas (both in the input and in the output) have one attribute. For each input area $v \in V$, the attribute value is denoted by γ_v. The problem definition requires that in each output region one of the contained input areas c is selected as a center, which dictates the attribute value of that output region as γ_c. This means that no "new" attribute values arise. Furthermore, the basic ILP (with additional constraints ensuring contiguity) suffices for area aggregation—if the assignment costs $a_{c,v}$ are appropriately set. Haunert and Wolff [HW10a] define $a_{c,v}$ to express two criteria.

First, since $x_{c,v} = 1$ implies that area v changes its attribute value (or *color*) from γ_v to γ_c and an objective is to keep such changes small, a cost f_{recolor} is charged that depends on

the *distance* from γ_v to γ_c. This distance is generally defined with a function $\delta\colon \Gamma^2 \to \mathbb{R}_{\geq 0}$, where Γ is the set of all attribute values. It is chosen to reflect the *dissimilarity* of the attribute values, which implies that by minimizing f_{recolor} the objective for grouping similar areas is addressed. The cost for color change is defined as

$$f_{\text{recolor}} = \sum_{c \in V} \sum_{v \in V} w(v) \cdot \delta(\gamma_c, \gamma_v) \cdot x_{c,v} \,. \tag{4.8}$$

Second, the objective for geometrically compact output regions is generally modeled with a cost $f_{\text{non-compact}}$. In this article, we define

$$f_{\text{non-compact}} = \sum_{c \in V} \sum_{v \in V} w(v) \cdot d(c, v) \cdot x_{c,v} \,, \tag{4.9}$$

where $d\colon V^2 \to \mathbb{R}_{\geq 0}$ is the Euclidean distance between the centroids of the areas in V.

The overall objective is to minimize

$$f = \alpha \cdot f_{\text{non-compact}} + (1 - \alpha) \cdot f_{\text{recolor}}, \tag{4.10}$$

where $\alpha \in [0, 1]$ is a parameter that weights the two criteria. Accordingly, we use Objective (4.1) with assignment costs

$$a_{c,v} = w(v) \cdot \Big(\alpha \cdot d(c, v) + (1 - \alpha) \cdot \delta(\gamma_c, \gamma_v) \Big) \,. \tag{4.11}$$

Haunert and Wolff [HW10a] assumed that Γ is a discrete set of land cover classes and that the distance δ measures the semantic dissimilarity of classes. In our example, however, we are given areas with unemployment rates and thus a quantitative attribute. More specifically, $\Gamma = [0, 1]$ is the set of real numbers between 0 and 1. We simply define the distance

$$\delta(\gamma_1, \gamma_2) = |\gamma_2 - \gamma_1| \quad \text{for} \quad \gamma_1, \gamma_2 \in \Gamma \,. \tag{4.12}$$

With this definition, the requirement that each region contains an area that does not change its attribute value has no influence on the optimal objective value. In particular, if all areas have the same weight, the cost f_{recolor} would be minimized by selecting the median of all attribute values in a region.

A problem that is similar to our problem variant of area aggregation with a quantitative attribute is the aggregation of 3D building models based on their heights. Guercke et al. [GGBS11] have approached this problem by integer linear programming, using a model that is similar to the model of Haunert and Wolff [HW10a].

4.3 Handling ILPs with large sets of constraints

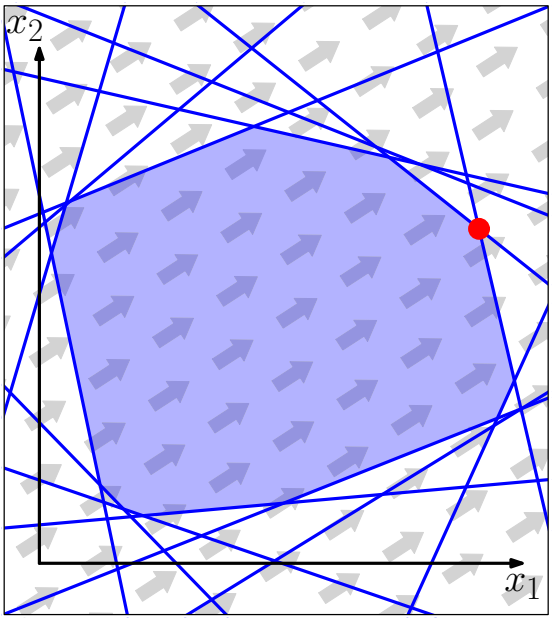
The cutting-plane method that we will present in this article is based on an ILP consisting of Objective (4.1), Constraints (4.2)–(4.5), and an exponential number of constraints that ensure contiguity, which we term *contiguity constraints*. Such a large set of constraints is

only reasonable with special constraint-handling techniques, which we sketch in this section. Generally, the strategy is to first disregard some constraints of the original model and to consider a reduced model that contains only few constraints ensuring some very basic properties of a solution. In our example, we disregard the contiguity constraints.

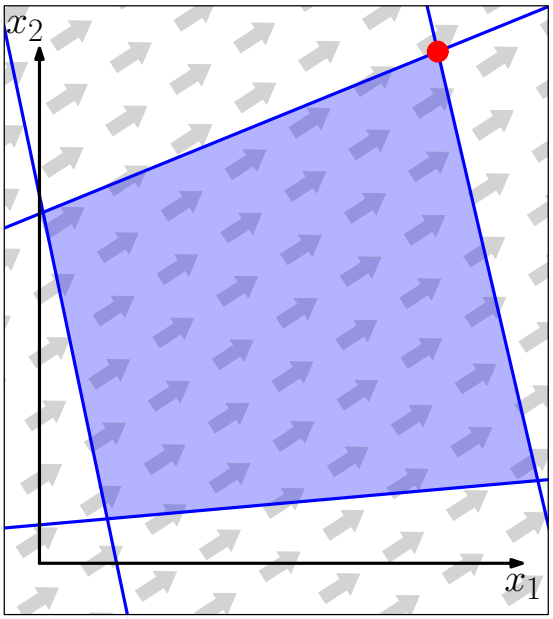
The most simple approach following this strategy is to first compute an optimal solution to the reduced model. Then, it is often easy to check whether the solution is also a solution to the original model. For example, non-contiguous regions can be easily and efficiently detected using a breadth-first search [CLR90]. Generally, if the solution satisfies all constraints of the original model, one is done. Else, one can augment the model with exactly those constraints that were found to be violated—and solve it again. This process is repeated until an optimal solution to the current model is found and asserted to satisfy all constraints of the original model. Drexl and Haase [DH99] and Duque [DCM11], for example, use this approach in order to solve the problem of sales force deployment and the p-regions problem, respectively. An advantage of this approach is that the ILP solver can still be thought of and used as a black box. That is, after each augmentation of the model with constraints, the ILP solver can be applied with default settings and without any intervention. Also, the approach can be more efficient than an approach with a complete instantiation of the model. On the other hand, solving the model to optimality after each augmentation step can be quite inefficient.

A better approach is to repeatedly augment the model but to avoid computing an optimal solution after each augmentation step. Instead, one can halt some ILP solvers as soon as they find a new *incumbent solution*, that is, an integral solution that satisfies all constraints of the current model and that is better than any such solution found so far. As in the most simple approach, this incumbent solution needs to be inspected using an algorithm that is specifically designed and implemented for that purpose, e.g., in the case of area aggregation, an algorithm that checks whether the regions corresponding to the solution are contiguous. If a constraint violation (e.g., a non-contiguous region) is found, a new constraint is added and the optimization process is resumed—rather than repeated from scratch. This kind of intervention (i.e., halting the solver, inspecting the incumbent solution, augmenting the model, and resuming the optimization process) can be done via interfaces specified in the programming libraries of solvers like CPLEX and Gurobi. As the most simple iterative approach, this approach finally yields a globally optimal solution. Examples for the application of this method in computational cartography are presented by Haunert and Wolff [HW10b] and Nöllenburg and Wolff [NW10].

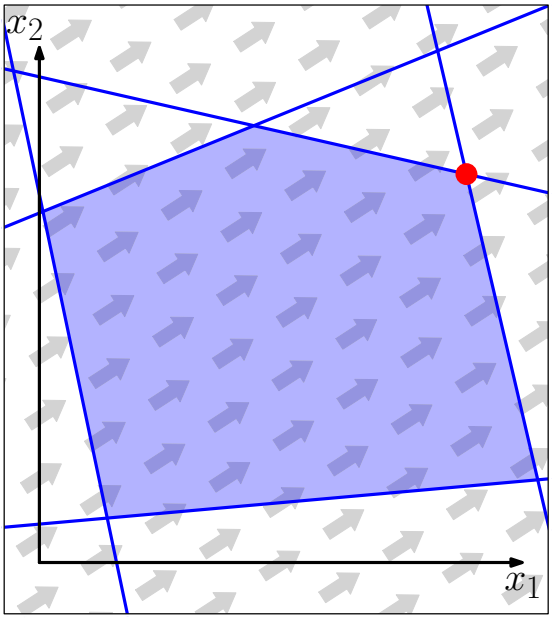
Finally, the approach that we apply in this article generates relevant constraints without relying on incumbent (i.e., integral) solutions to the ILP. This is advantageous because finding an integral solution can be just as hard as finding an optimal solution to a model and, therefore, may take very long. Instead of inspecting incumbent solutions for constraint violation, we inspect optimal solutions of *LP relaxations*—in the LP relaxation corresponding to a model the variables are allowed to take fractional values, which makes the problem less complex, even efficiently solvable [Kar84]. Most ILP solvers begin with solving the LP relaxation of the given model to find a lower bound (or upper bound in case of a maximization problem) of the objective function, usually by applying a variant of the simplex algorithm [Dan63], which is fast in practice. Furthermore, during the optimization process,



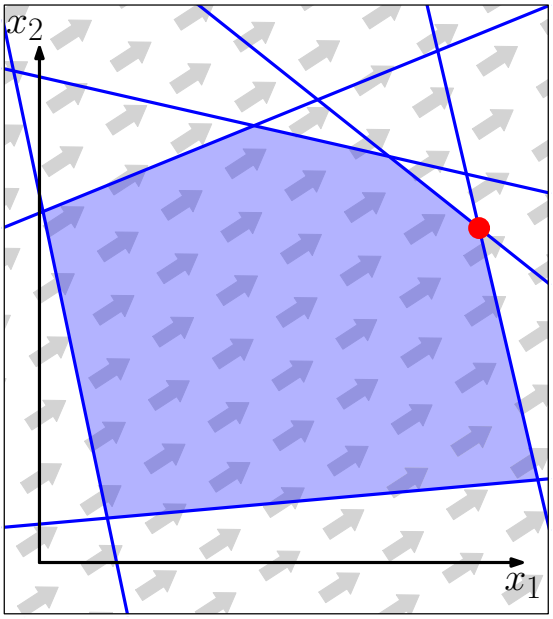
(a) The LP with all constraints.



(b) The LP with a selected subset of constraints.



(c) The LP with an enhanced subset of constraints.



(d) The LP after all the necessary separation steps (here: two).

Figure 4.3: A geometric interpretation of an LP with two variables $x_1, x_2 \in \mathbb{R}$ (constraints presented as blue lines, objective function presented as gray arrows) and its solution (red). While (a) depicts the complete LP and an optimal solution, (b) to (d) show the advantages of using a separation algorithm: Starting with only a subset of constraints (b) we extend the LP with constraints which are violated by the solution to the current LP (c) as long as we can find any (see Fig. 4.4). When the separation algorithm cannot find any violations (d) the current solution is valid and optimal with respect to the complete LP of the original model.

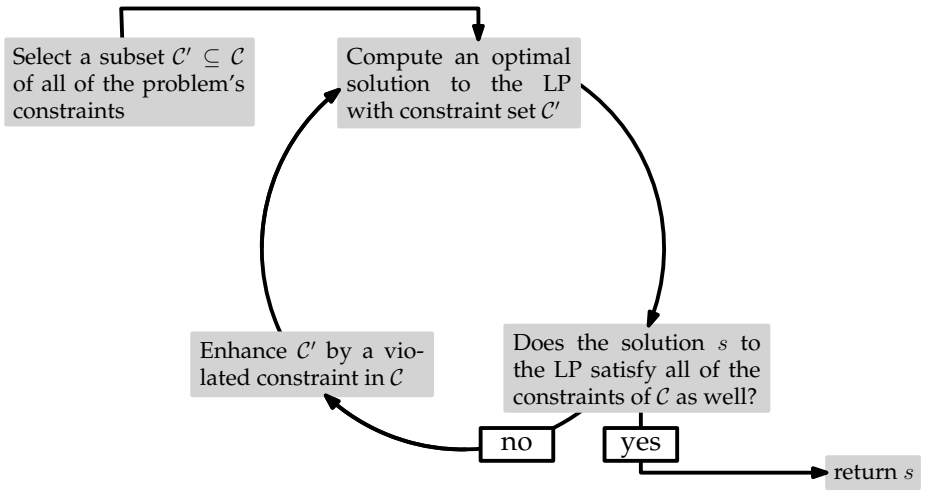


Figure 4.4: The way from Fig. 4.3(b) to (d) as a diagram; the separation algorithm leads to a point of diversion depending on whether a violated constraint exists.

most solvers also solve LP relaxations for sub-instances (i.e., branch-and-bound nodes) in which the values of some variables are fixed. In any case, we test whether a solution of an LP relaxation of the current model satisfies all constraints of the original model. More precisely, we use a *separation algorithm*, that is, an algorithm that either asserts that all constraints are satisfied or yields a violated constraint, with which we then augment the model (see Figs. 4.3 and 4.4). For the Traveling Salesman Problem[1], for example, this approach outperforms other ILP approaches [Pat03]. In comparison to the previous approach based on incumbent solutions, one avoids an extensive exploration of the branch-and-bound tree. On the other hand, designing a separation algorithm can be a non-trivial task. If the constraints are of a certain type, the separation step can be done efficiently (in particular, without explicitly testing all inequalities), for example by computing a maximum flow in an appropriately defined graph. Carvajal et al. [CCG+13] deal with forest planning models using this method. In Section 4.4 we exemplify this approach in detail for area aggregation.

4.4 A new method for area aggregation using cutting planes

In Section 4.2 we have modeled all aspects of the problem that we aim to solve. That is, we consider Objective (4.1) with the assignment costs in Equation (4.11) and the definition of the distance δ in Equation (4.12). We minimize this objective subject to Constraints (4.2)–(4.5) and constraints ensuring contiguity. Obviously, we could ensure contiguity with Constraints (4.6) and (4.7), but in this section we introduce an alternative formulation that we use with our cutting-plane method. An advantage of this formulation is that we get along with the binary variables $x_{c,v}$ and without the additional variables $y^c_{(u,v)}$.

[1] Dealing with the Traveling Salesman Problem means determining a shortest path visiting every city of a given set of cities. It serves as a basis for various common GI science problems like one of its extensions, the vehicle routing problem [Lap92, Chr76a, BN68].

Instead of setting up the model completely, we let the solver begin with the basic ILP from Section 4.2.1. When the solution to the LP relaxation in a branch-and-bound node is found, we intervene and check whether some of the not yet added contiguity constraints presented in the following are violated and add at least one of them in that eventuality.

With the following constraints we ensure contiguity. They are inspired by the work of Carvajal et al. [CCG+13] and Drexl and Haase [DH99], where they were set up in order to solve different problems: forest planning and sales territory alignment. The algorithm of Carvajal et al. solves a problem asking for the selection of a single output region and is therefore not directly applicable to the area aggregation problem. Drexl and Haase deal with districting but add violated constraints based not on the solution to the LP relaxation in a branch-and-bound node but to an optimal solution to the current model. Afterwards, they run experiments to calculate only upper and lower bounds for larger instances, but no optimal solutions. We do not only present these constraints in combination but also emphasize the advantages of cutting-plane methods. Furthermore, we contribute an ILP formulation for area aggregation.

4.4.1 Constraints completing the ILP formulation

In the following, we present two different kinds of constraints. While combining the basic ILP from Section 4.2.1 with the constraints presented first in this section is necessary and sufficient for area aggregation, the constraints presented afterwards work in a supporting way. These supportive constraints mainly enforce the minimum-weight constraints (Constr. (4.4)) in a more determined manner.

Contiguity constraints based on vertex separators

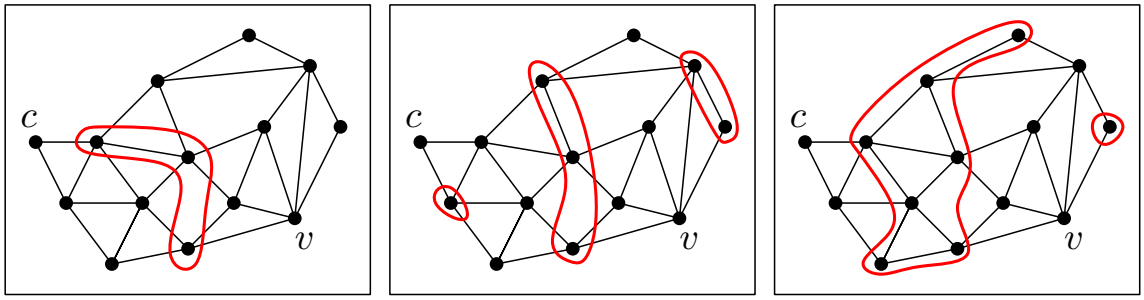
Let $c, v \in V$ be two arbitrary areas. In the following we consider c the center of a region and the possibility of assigning v to this region, which is represented with the binary variable $x_{c,v}$.

Furthermore, let $\mathscr{S}_{c,v} \subseteq 2^V$ be the set of all (c,v)-*separators* in G, where a set $S \subseteq V \setminus \{c,v\}$ is called a (c,v)-separator if every path from c to v in G contains at least one vertex in S (see Fig. 4.5). If v is allocated to the region with center c, then—for the sake of contiguity—for each (c,v)-separator $S \in \mathscr{S}_{c,v}$ at least one area of S has to be allocated to this region as well. In linear terms this condition is expressed as follows:

$$\sum_{u \in S} x_{c,u} \geq x_{c,v} \quad \text{for each } S \in \mathscr{S}_{c,v},\ c,v \in V. \tag{4.13}$$

If $c = v$ or $\{c,v\} \in E$, that is, areas c and v are identical or adjacent, the set of (c,v)-separators $\mathscr{S}_{c,v}$ is empty. Consequently, there is no constraint described in Formula (4.13) for these cases. In general, the number of separators is in $\mathcal{O}(2^{|V|})$.

The basic model (see Section 4.2.1) with the constraints presented here is already fully operative: All output regions are contiguous if and only if none of the constraints based



(a) (b) (c)

Figure 4.5: An example of an adjacency graph G with various (c, v)-separators (red): Every path from v to c contains at least one vertex of the respective separator.

on vertex separators is violated. To see why, we only consider the case where the right-hand side equals 1, since otherwise the inequalities from Constraint (4.13) are always true. Suppose an arbitrary region containing a center c and another area v is contiguous. This means a path P between c and v exists in G that consists only of vertices belonging to the same region. Since each separator $S \in \mathscr{S}_{c,v}$ contains at least one vertex of any path from c to v, it also contains a vertex of path P and thus an area of the region in question. Consequently, the inequalities from Constraint (4.13) are fulfilled. Now, let us assume that a region $R \subseteq V$ is not contiguous, i.e., R consists of at least two connected components. Thus, it is possible to take a look at the region's center c and an area v contained in another connected component than c. Then the set $V \setminus R$ is a (c, v)-separator and the inequality from Constraint (4.13) is violated for $S = V \setminus R \in \mathscr{S}_{c,v}$.

Supportive constraints inspired by the minimum-weight requirement

If a separator $S \in \mathscr{S}_{c,v}$ has certain properties, we define another constraint: Since S is a (c, v)-separator, the set $V \setminus S$ consists of at least two connected components—one of which contains c and one of which contains v. Let $C(S, c)$ be the connected component in $V \setminus S$ containing c. If the total weight of $C(S, c)$, that is $\sum_{u \in C(S,c)} w(u)$, does not exceed the minimum weight $w_{\min}$ demanded for a resulting region, then every region with center c has to contain at least one area of the separator S. Or stated as a linear inequality:

$$\sum_{u \in S} x_{c,u} \geq x_{c,c} \quad \text{for each } S \in \mathscr{S}_{c,v} \text{ with } \sum_{u \in C(S,c)} w(u) < w_{\min},\ c, v \in V \tag{4.14}$$

Again, the inequality is always fulfilled for $x_{c,c} = 0$. Only if $x_{c,c} = 1$ (i.e., c is declared a center) the allocation of at least one area in the respective separator is required.

Constraint (4.14) alone does not suffice to achieve contiguity. When combining it with Constraint (4.13), however, it acts in a supportive manner. To see why, we contrast the contiguity constraint (Constr. (4.13)) with the supportive constraint (Constr. (4.14)). We observe that they only differ with respect to their right-hand sides, which are $x_{c,v}$ and $x_{c,c}$, respectively. Constraint (4.3) ensures $x_{c,c} \geq x_{c,v}$ and, thus, Constraint (4.14) is at least as

restrictive as Constraint (4.13) for any $v \in V$. That is, every solution to an LP relaxation considering Constraints (4.3) and (4.14) fulfills Constraint (4.13). However, the solution to an LP relaxation considering Constraints (4.3) and (4.13) cannot give this guarantee with respect to Constraint (4.14). Transferring this observation to Fig. 4.3, we note that a more restrictive constraint cuts away more parts of the solution polytope (marked as a blue region in the figures). This results in a better performance of solvers based on branch and bound [NW88].

4.4.2 Adding the constraints

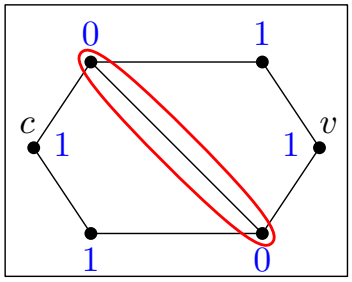
As described in Section 4.3, the ILP is built up step by step. In this section, we describe a separation algorithm, that is, an algorithm that allows us to find at least one violated contiguity constraint if there exists any. Then, we show how such a contiguity constraint may also provide a supportive constraint. Afterwards, we discuss how to speed up the computation by restricting the search for violated constraints, and we argue why this does not harm the correctness. Finally, we present another method that may find additional violated constraints without much computational overhead.

A pseudocode formulation of these algorithms can be found in Section 4.A.

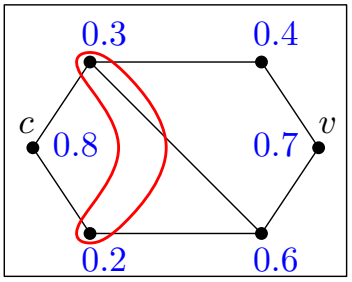
Finding violated contiguity constraints using minimum-weight vertex separators

If we assume that the integrality constraints $x_{c,v} \in \{0, 1\}$ are satisfied for all variables (see Fig. 4.6(a)), finding a violation of Constraint (4.13) is straightforward. For example, for a fixed center c, we could consider the subgraph of G induced by all nodes v with $x_{c,v} = 1$. If c and a node v lie in different connected components of this graph, a violation of a contiguity constraint exists. However, since we want to find constraints in a preliminary stage of the solution (i.e., in the solution of the LP relaxation of the ILP), we need to deal with variables of fractional values (see Fig. 4.6(b)).

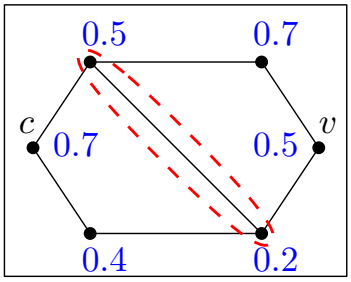
To find contiguity constraints that are violated by the solution to the current LP relaxation, we proceed as follows (see also Algorithm 13 in 4.A): Taking a closer look at Constraint (4.13), we observe that separators providing a smaller sum on the constraint's left-hand side are more likely to implicate a violation of the corresponding constraint, since the right-hand side does not depend on the choice of the separator. For every potential center c, we therefore take for every $v \in V$ the value $x_{c,v}$ of the current solution as the weight of v in G and focus on *minimum-weight* (c, v)-separators afterward—in Section 2.2.3, *Application: Minimum-Weight Vertex Separators*, we describe how to compute minimum-weight vertex separators by using minimum edge cut algorithms [CLR90, EK72, FJF56a]. The reason why this is effective is the following: If the inequality from Constraint (4.13) is violated for an arbitrary (c, v)-separator S, it is also violated for a minimum-weight (c, v)-separator S^* since $x_{c,c} > \sum_{u \in S} x_{c,u} \geq \sum_{u \in S^*} x_{c,u}$ holds in that case. Thus, for specific $c, v \in V$, looking at a minimum-weight (c, v)-separator guarantees to notice a violation if there is one (see Fig. 4.6(c)). In particular, computing and examining minimum-weight (c, v)-separators for every potential center $c \in V$ and every $v \in V \setminus \{c\}$ solves the separation problem.



(a) With integral $x_{c,u}$-values, it is easy to find a vertex separator, for which the contiguity constraint is violated.
$0 + 0 = \sum_{u \in S} x_{c,u} < x_{c,v} = 1$



(b) For fractional $x_{c,u}$-values, a violation of a contiguity constraint is harder to find. A violation exists, for example, for the depicted separator.
$0.3 + 0.2 = \sum_{u \in S} x_{c,u} < x_{c,v} = 0.7$



(c) The contiguity constraint is satisfied for the minimum-weight (c, v)-separator. Thus, no contiguity constraint is violated for any (c, v)-separator.
$0.5 + 0.2 = \sum_{u \in S} x_{c,u} \geq x_{c,v} = 0.5$

Figure 4.6: In this example with fixed $c, v \in V$, we examine the search for a (c, v)-separator, for which the contiguity constraint (Constr. (4.13)) is violated. For a vertex $u \in V$, we consider the value $x_{c,u}$ from the solution of an LP relaxation as its vertex weight (blue). It depends on these values whether a violation of a contiguity constraint exists. Separators which are interesting in this context are indicated in red.

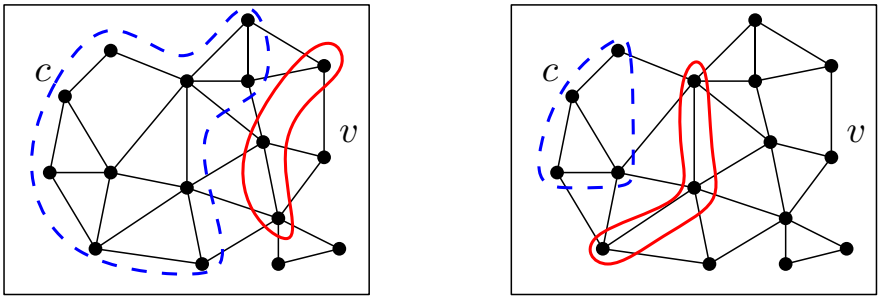
Finding supportive constraints

For each vertex computed separator, we proceed as follows to find a supportive constraint. The vertex separator splits V into at least two connected components in G. If one of these components forms a region with less than the minimum weight, the corresponding separator offers Constraint (4.14) for any potential center in that component. Therefore, we check for every area in the connected component of c whether the respective Constraint (4.14) is violated for this area as a center. If that is the case, we add the violated constraint to the model. This means we compute a separator principally for a certain center c but also use it to set up constraints independent from c.

The algorithm yields a minimum-weight (c, v)-separator. Such a separator is not unique. Among the minimum-weight (c, v)-separators we prefer those which are closer to c (see Fig. 4.7). The reason why we prefer separators closer to c is Constraint (4.14): That way, it is more likely to detect violations of Constraint (4.14) and consequently more likely to add a supportive constraint. In order to find the minimum-weight (c, v)-separator closest to c we use the fact that our algorithm is based on the search for a minimum edge cut. Here, common algorithms follow the same scheme and return the minimum edge cut closest to a certain vertex [CLR90].

Restricting the search for violated constraints

A straightforward implementation of the algorithm implicates the computation of a quadratic number of vertex separators every time an LP relaxation is solved optimally. In order to reduce this number, we make restrictions described in the following.



(a) (b)

Figure 4.7: The (c, v)-separator (red) divides the graph into at least two connected components. The component of c (dashed, blue) in (b) is more likely to offer a Constraint (4.14) than the one in (a).

We read the value of a variable $x_{c,v}$ for arbitrary $c, v \in V$ as the tendency of v to be allocated to c. With regard to Constraint (4.2), we see that, for a fixed $v \in V$, the average value of the variables $x_{c,v}$ is $\frac{1}{|V|}$. Therefore, we interpret $x_{c,v} \geq \frac{1}{|V|}$ as an indicator that allocating v to c (or, in the case $v = c$, declaring v a center) is—for the moment—considered reasonable. Conversely, a value less than $\frac{1}{|V|}$ implicates that an allocation to a different center is preferable. Hence, we impose the following restrictions on the method presented in this section:

- We compute minimum-weight (c, v)-separators only for areas v with $x_{c,v} \geq \frac{1}{|V|}$.
- We take centers c with $x_{c,c} < \frac{1}{|V|}$ only into consideration if no violation is detected for more probable centers.
- We add a supportive constraint (Constr. (4.14)) only for areas v in the connected component (not fulfilling the minimum-size requirement) of the examined center for which $x_{v,v} \geq \frac{1}{|V|}$ holds.

Although we fail to notice violations of the inequality from Constraint (4.13) for areas with $x_{c,v} < \frac{1}{|V|}$ and, thus, to solve the separation problem properly, this fact does not pose a risk to finding a contiguous solution. As we handle these violations in a branch-and-bound node, the solver goes on branching and bounding in the node's sub-instances. It continues until either an integral solution is found or the problem becomes infeasible. If a sub-instance is infeasible, there is no need to worry about whether the number of added constraints is too small. In case of an integral solution, there are two possible scenarios for each potential region with center c: Either $x_{c,c} = 0$ and with it $x_{c,v} \leq x_{c,c} = 0$ for every $v \in V$ (see Constraint (4.3)), i.e., the region is not considered in the solution and therefore in no need of a validation of contiguity. Or $x_{c,c} = 1 \geq \frac{1}{|V|}$, which results in a verification of the region's contiguity and implies that we find violations if existing.

Finding violated contiguity constraints using connected components

For the following method of finding a useful vertex separator (see also Algorithm 14 in 4.A), we interpret for every probable center c (i.e., $x_{c,c} \geq \frac{1}{|V|}$) the values of the variables $x_{c,v}$ for

$v \in V$ as described above, only stricter: If $x_{c,v} \geq \frac{1}{|V|}$, we consider v allocated to the region with center c, otherwise we do not. Therefore, we take a look at

$$V' = \left\{ v \in V \,\middle|\, x_{c,v} \geq \tfrac{1}{|V|} \right\}$$

and to the subgraph of G induced by V'. In this subgraph we are able to detect connected components of the areas tending to be allocated to c. For every connected component $U \subseteq V' \subseteq V$ with $c \notin U$, we examine the set of adjacent areas in V, that is

$$B_U \coloneqq \left\{ w \in V \setminus U \,\middle|\, \exists_{u \in U} : \{u, w\} \in E \right\}, \tag{4.15}$$

and take it as a vertex separator. Subsequently, we add a Constraint (4.13) for every $u \in U$ (and c). This way we have a good chance of finding additional violated contiguity constraints without much computational overhead.

4.5 Results and discussion

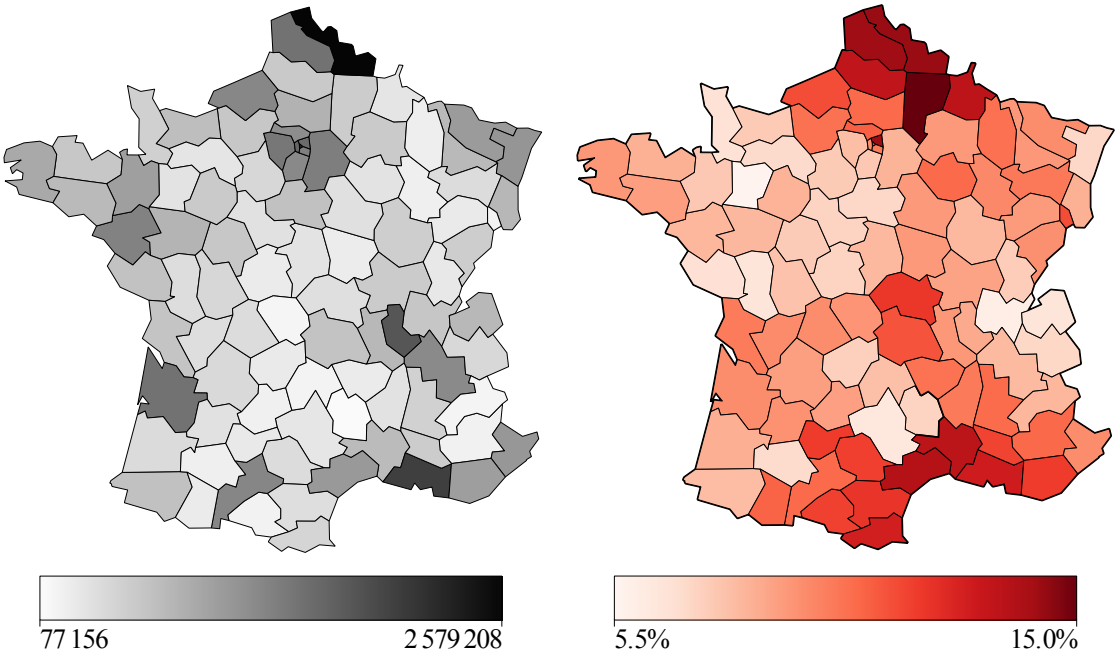
For a series of experiments we used a computer with a 3.2 GHz Intel Core i5-4570 Processor and 16 GB RAM. Our program is written in Java and besides the Gurobi [Gur18] interface for Java, we use in particular the library JGraphT [Nav16] for building the adjacency graph and performing operations on it (see also 4.A). The data for the experiments is provided by the European statistical service (Eurostat, [Eur15]) and the European Observation Network for Territorial Development and Cohesion (ESPON, [Eur11]). This data is gathered with regard to the NUTS[2] subdivision of the European Union. As an example, we examine the unemployment rate [Eur11] for the NUTS-3 subdivision of continental France (see Fig. 4.8(b)) which means processing a subdivision of 94 departments. In contrast to Haunert and Wolff's original work [Hau08, HW10a] we take the population of a department [Eur15] (see Fig. 4.8(a)) instead of its area as the department's weight. This means we compare the methods described in Sections 4.2.2 and 4.4 on the third level administrative subdivision of a major European country. We do not only compare the effects of α, the factor weighting the cost functions f_{recolor} and $f_{\text{non-compact}}$ in the objective function, on the result and the computation time, but also examine the influence of the choice of $w_{\min}$, the minimum weight required of a region in the output.

As attribute domain, $\Gamma = [0, 1]$ is given and we write $\gamma_v \in \Gamma$ for the attribute value, i.e., unemployment rate, of an area $v \in V$. According to Section 4.2.3, we define $\delta(\gamma_1, \gamma_2) = |\gamma_1 - \gamma_2|$ for arbitrary $\gamma_1, \gamma_2 \in \Gamma$ as the cost for recoloring. The cost for non-compactness is defined through the Euclidean distance d between centroids. Observing Equations (4.1) and (4.8)–(4.10) we get the following overall cost function as the objective function:

$$\sum_{c \in V} \sum_{v \in V \setminus \{c\}} w(v) \cdot \Big(\alpha \cdot d(c, v) + (1 - \alpha) \cdot \delta(\gamma_c, \gamma_v) \Big) \cdot x_{c,v} \tag{4.16}$$

As for the weighting factor α, we make the following choice: With $\alpha \in \{0, 1\}$ we want to present the extreme solutions. For $\alpha = 1$ we receive compact regions of a certain weight

[2] Nomenclature des unités territoriales statistiques



(a) Population (department weight).

(b) Unemployment rate (attribute value; origin of data: EUROSTAT, ESPON estimations, 2011).

Figure 4.8: input data for statistical aggregation of France's departments

ignoring any statistical similarities between the areas. For $\alpha = 0$ we see a solution with least recoloring costs but consisting of regions of mostly unconventional shape. According to the experiments that we present in this section, a weighting factor of $\alpha = 2 \cdot 10^{-5}$ still yields almost optimally compact shapes and $\alpha = 1.25 \cdot 10^{-6}$ almost optimally homogenous regions. Therefore, we particularly tested our method for $\alpha = 0$ and $\alpha = 1$ as well as for $1.25 \cdot 10^{-6} \leq \alpha \leq 2 \cdot 10^{-5}$.

In our experiments, we set $w_{\min}$ to 5 % and 10 % respectively of the total population of continental France. With this setting, we end up with a partitioning into a number of larger regions which is approximately at the same scale as the actual NUTS-2 subdivision of continental France, which consists of 12 regions.

Considering the results in Tables (1) and (2), it comes to attention that our cutting-plane approach outperforms the flow model in every instance. In our experiments, the flow MILP is competitive only in the situation where $w_{\min}$ is 10 % of the total population and $\alpha = 1$. In every other case, our ILP is many times faster, as the fifth column of each table indicates. With α declining, i.e., focusing on reducing f_{recolor}, the advantages of the cutting-plane formulation become clearer since the problem becomes harder to solve. This is caused by

[3] Since the calculations are incomplete for both the flow model and the cutting-plane approach, the result presented here is the best one found with no guarantee of optimality. It is the result of five days of calculation with the cutting-plane approach. The optimality gap is 6.89 %, i.e., the resulting cost of an optimal solution is at least $1 - 0.0689 = 0.9311$ times the cost of the solution presented here. (Using the flow MILP, the remaining gap is 51.4 %.)

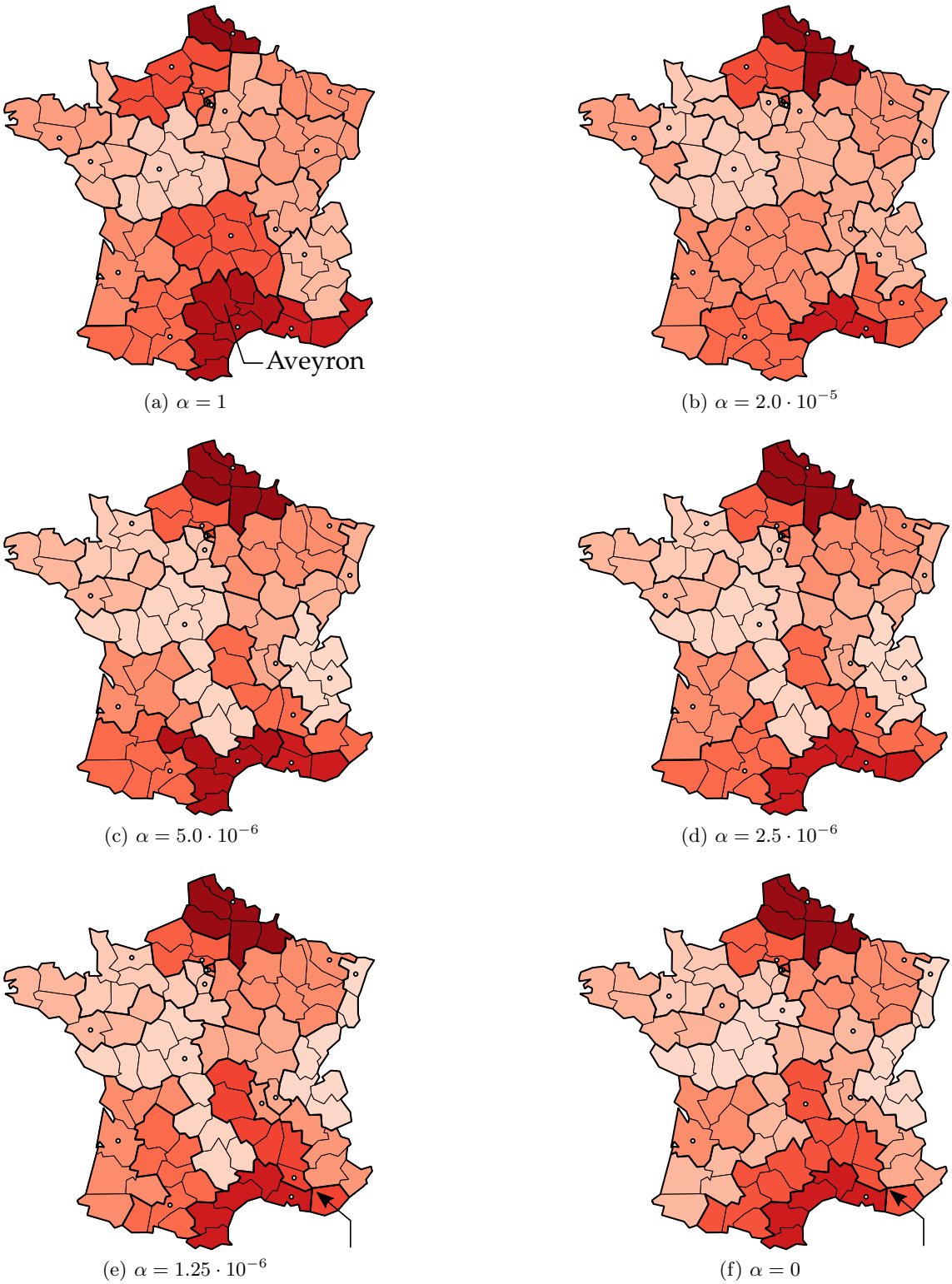
Fig. 4.9	α	t_{cut}	t_{flow}	$t_{\text{flow}}/t_{\text{cut}}$	$\#_{\text{out}}$
(a)	1	30 s	497 s	16.6	18
(b)	$2.0 \cdot 10^{-5}$	21 s	502 s	23.9	18
(c)	$5.0 \cdot 10^{-6}$	50 s	14 229 s ≈ 4.0 h	284.6	17
(d)	$2.5 \cdot 10^{-6}$	100 s	> 5 d	> 4 320.0	16
(e)	$1.25 \cdot 10^{-6}$	451 s	> 5 d	> 957.9	16
(f)	0	18 413 s ≈ 5.1 h	> 5 d	> 23.4	14

(1) Running times for the experiments with minimum weight $w_{\min} = \frac{1}{20} \sum_{v \in V} w(v)$.

Fig. 4.10	α	t_{cut}	t_{flow}	$t_{\text{flow}}/t_{\text{cut}}$	$\#_{\text{out}}$
(a)	1	20 s	35 s	1.8	9
(b)	$2.0 \cdot 10^{-5}$	59 s	267 s	4.5	9
(c)	$5.0 \cdot 10^{-6}$	194 s	14 464 s ≈ 4.0 h	74.6	9
(d)	$2.5 \cdot 10^{-6}$	495 s	> 5 d	> 872.7	9
(e)	$1.25 \cdot 10^{-6}$	86 981 s ≈ 1 d	> 5 d	> 4.9	9
(f)	0	> 5 d	> 5 d	—	g [3]

(2) Running times for the experiments with minimum weight $w_{\min} = \frac{1}{10} \sum_{v \in V} w(v)$.

Tables (1) and (2): These tables provide information about the running times in seconds (s), hours (h) or days (d) for the corresponding experiments with different minimum weights (see Eq. (4.4)). Experiments were aborted after five days if no optimal solution was found; this is indicated with "> 5 d." The first column refers to the visual presentation of the result. The second column denotes the α-value used in this experiment (see Eq. (4.16)). In the column marked with t_{cut} one can find the running times of our approach described in Section 4.4, in the column marked with t_{flow} the times of the state-of-the-art approach of Section 4.2. The fifth column compares those values by giving the ratio $t_{\text{flow}}/t_{\text{cut}}$. Finally, the column marked with $\#_{\text{out}}$ gives additional information about the structure of the result by providing the number of resulting regions.



(a) $\alpha = 1$

(b) $\alpha = 2.0 \cdot 10^{-5}$

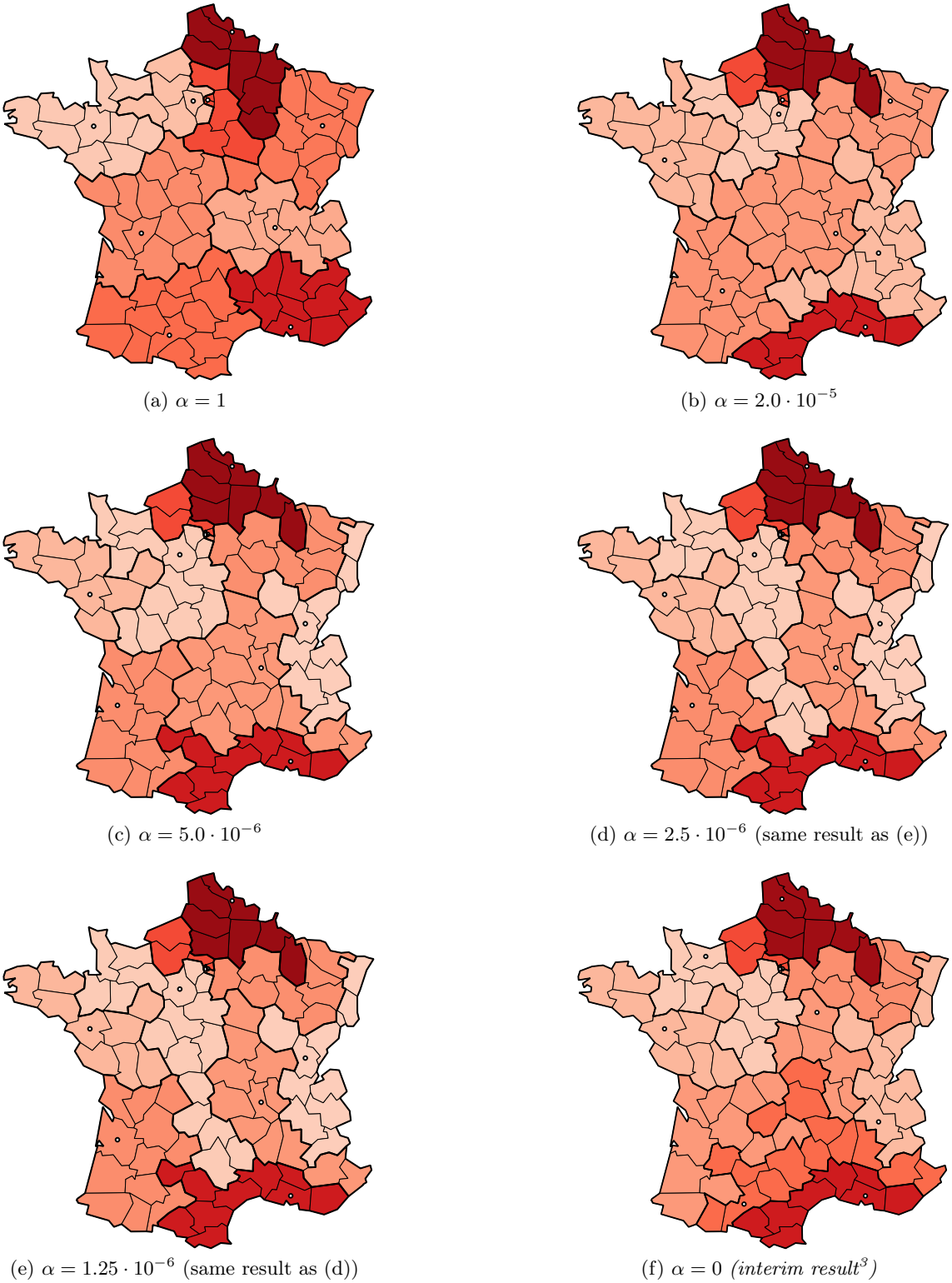
(c) $\alpha = 5.0 \cdot 10^{-6}$

(d) $\alpha = 2.5 \cdot 10^{-6}$

(e) $\alpha = 1.25 \cdot 10^{-6}$

(f) $\alpha = 0$

Figure 4.9: Output with $w_{\min}$ equaling 5 % of France's total population; thick lines mark aggregated regions colored according to the unemployment rate of their center (∘). The arrows in Fig. (e) and (f) point to extremely narrow parts of contiguous regions.



(a) $\alpha = 1$

(b) $\alpha = 2.0 \cdot 10^{-5}$

(c) $\alpha = 5.0 \cdot 10^{-6}$

(d) $\alpha = 2.5 \cdot 10^{-6}$ (same result as (e))

(e) $\alpha = 1.25 \cdot 10^{-6}$ (same result as (d))

(f) $\alpha = 0$ *(interim result[3])*

Figure 4.10: Output with $w_{\min}$ equaling 10 % of France's total population; thick lines mark aggregated regions colored according to the unemployment rate of their center (∘).

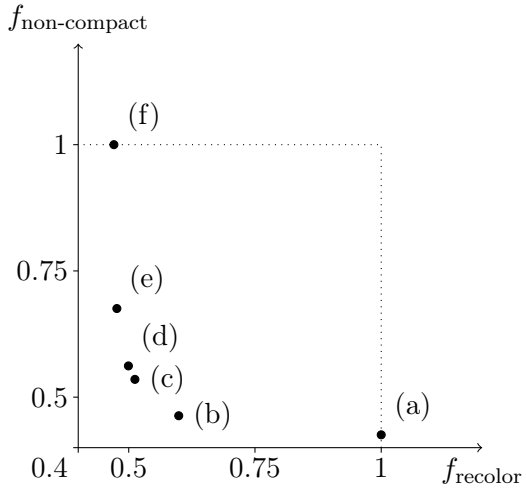
the fact that focusing on compactness supports contiguity: A compact region is more likely to be contiguous than a region of areas with similar attributes.

In particular, the three last rows of each table deserve our special attention. For both settings of $w_{\min}$, the solver is not able to return a result using the flow model for $\alpha \leq 2.5 \cdot 10^{-6}$. Leaving the instance with $\alpha = 0$, $w_{\min} = \frac{1}{10} \sum_{v \in V} w(v)$ out of account, each of these instances is solvable with the cutting-plane algorithm. For three of these instances, the solver returns an optimal solution within a few minutes.

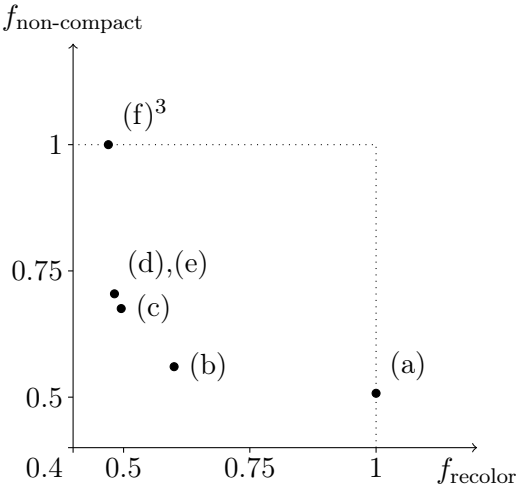
Considering the output in detail, we find that the noticeable higher unemployment rates in northern and southern France (former Nord-Pas-de-Calais and on the Mediterranean coast) are identified and aggregated in every example where similarity is considered (i.e., $\alpha \neq 1$). In general, for most of the departments, the resulting color (unemployment rate) is close to the input. Nevertheless, one also notices larger differences between input and output coloring for various departments. This phenomenon occurs especially for $\alpha = 1$, but also in other cases. For $\alpha \neq 1$, however, this only causes minor changes since these departments have a comparatively small population (see Fig. 4.8(a)) and consequently contribute only little cost. Take, for example, Aveyron, a department in south-central France (see Fig. 4.9(a)) with one of the lowest unemployment rates of all departments. For both $\alpha = 1$ and $\alpha = 0$ it has undergone seemingly expensive recoloring. Considering the objective, this makes sense as less than 0.5 % of France's total population lives in Aveyron.

Another negative aspect is the fact that parts of some of the resulting regions are very narrow. Understandably, this occurs especially for $\alpha = 0$, where resulting regions reach diagonally from one border to another, e.g., in both Figs. 4.9(f) and 4.10(f) the region containing the department in the very south-west (Pyrénées-Atlantiques). For $\alpha \neq 0$ this phenomenon occurs less distinctly. A problem that arises independent of α is bottlenecks. A bottleneck is a very narrow part of a region connecting two larger parts. In order to resolve this, one has to manipulate the input data or rather its interpretation. Building the adjacency graph G, we consider departments adjacent as soon as they share a boundary. When dealing with a map of France's departments, we have to handle several pairs of departments sharing a borderline of less than 10 km. Without adjusting the adjacency rule, we see results with constellations such as in Figs. 4.9(e) and 4.9(f), where a region bordering the Mediterranean Sea (marked with an arrow in the south-east) even seems to consist of two components. Here, the departments Var and Vaucluse share less than 1 km of borderline.

Figure 4.11 gives us additional arguments to disqualify $\alpha = 0$ or $\alpha = 1$ as reasonable weight factors. Let us take a look at the situation for $w_{\min}$ equaling 5 % of the total population (i.e., Fig. 4.11(a)). As $\alpha = 0$ (here: (f)) results in an optimal solution with respect to the cost for recoloring, these costs are higher for every other value of α. But for $\alpha = 1.25 \cdot 10^{-6}$ (here: (e)), recoloring is only approximately 1.2 % more expensive (47.06 % versus 47.65 % of the maximum value) whereas the cost for non-compactness decreases by 32.4 %. The situation is the same for the other value of $w_{\min}$ and similar with regard to the cost for non-compactness. Also, this argument supports our choice for $\alpha \approx 10^{-5}$. In this range, we get near-optimal results for either cost without unreasonable expenses for the other.



(a) Comparison for $w_{\min}$ equaling 5 % of France's total population, see Table (1) and Fig. 4.9; absolute maximum values are $8.0 \cdot 10^{12}$ (non-compactness) and $6.4 \cdot 10^{7}$ (recoloring).



(b) Comparison for $w_{\min}$ equaling 10 % of France's total population, see Table (2) and Fig. 4.10; absolute maximum values are $9.2 \cdot 10^{12}$ (non-compactness) and $8.4 \cdot 10^{7}$ (recoloring).

Figure 4.11: Costs for non-compactness and recoloring as a percentage of the maximum occurring cost; as 100 % of costs for non-compactness we take those arising while minimizing the cost for recoloring (i.e., $\alpha = 0$) and vice versa.

4.6 Conclusion

Our method allows us to solve instances as large as France's third level NUTS division in a reasonable amount of time, which the existing method based on a flow model does not. With our cutting-plane approach, we are able to partition about a hundred areas into nine to eighteen regions multiple times faster than with the compact flow model. In our experiments, the problem becomes harder the fewer output regions are asked for, as Tables (1) and (2) show. With increasing the number of areas of the input, our algorithm reaches its limits as well. Applying our model to Germany's third level division—with 402 districts the largest in the European Union—the solver returns no optimal solution within five days. This problem occurs even for $\alpha = 1$, i.e., focusing on compactness only.

Nevertheless, our approach offers unprecedented opportunities. Five of the instances from our experiments are set up too hard to be solved with the flow MILP within five days but are solvable with the cutting-plane algorithm within one day. In particular, the solver is capable to find an optimal solution for each of these settings which consider not only recoloring but also compactness, i.e., $\alpha \neq 0$. Moreover, three of these four cases are solved within only a few minutes. Due to these positive results, we argue that we have increased the range of problem instances that can be solved with proof of optimality such that it includes use cases that are of relevance. In particular, we consider it promising to use our method to produce benchmark solutions and to compare such solutions with solutions of efficient heuristic methods. Such a comparison may help to decide whether or not a heuristic

is a justifiable choice. Therefore, we think that both exact and heuristic methods for area aggregation can complement each other and will play their roles in the future.

In future work, we plan to support the ILP solver by calling heuristics during branching or in order to generate an initial solution. Furthermore, we consider it promising to apply our cutting-plane approach to the p-region problem [DCM11], in which the number of output regions is prescribed.

Appendix

4.A Appendix: Algorithms

In the following section we present the algorithms corresponding to the methods of adding constraints to the model which are described in Sect. 4.4.2. For both algorithms 13 and 14, we use—in addition to the adjacency graph $G = (V, E)$—a matrix of ILP variables indexed in the same manner as throughout the article ($x[c][v]$ corresponds to $x_{c,v}$). As already mentioned, we only consider x-values exceeding a certain bound; in Sect. 4.4.2 we choose $\frac{1}{|V|}$ which we use in these algorithms.

Algorithm 13: addVertexSeparatorCuts

Data: Adjacency graph $G = (V, E)$, V sorted such that $v \geq w \Leftrightarrow x_{v,v} \geq x_{w,w}$ for $v, w \in V$, Variable[ ][ ] x

Result: Cuts added by means of vertex separators (see Section 4.4.2)

```
1  Gen ← VertexSeparatorGenerator(G);
2  c ← V.first();
3  hasAdditionalCuts ← false;
4  while x[c][c].getValue() ≥ 1/V.size() ∨ ¬hasAdditionalCuts do
5      Gen.setWeights(V, x[c].getValues());
6      U ← { v ∈ V | v ≠ c ∧ (v, c) ∉ E ∧ x_{c,v} ≥ 1/V.size() };
7      foreach u ∈ U do
8          S ← Gen.getVertexSeparator(c, u);
9          if Σ_{v∈S} x[c][v].getValue() < x[c][u].getValue() then
10             addCut("Σ_{v∈S} x[c][v] ≥ x[c][u]");
11             hasAdditionalCuts ← true;
12         C ← connectivityInspector((V \ S, E|_{V\S})).getConnectedComponent(c);
13         if Σ_{v∈C} w(v) < w_min then
14             foreach v ∈ C with x[v][v].getValue() ≥ 1/V.size() do
15                 if Σ_{u∈S} x[v][u].getValue() < x[v][v].getValue() then
16                     addCut("Σ_{i∈S} x[v][u] ≥ x[v][v]");
17                     hasAdditionalCuts ← true;
18     c = V.next();
19 return hasAdditionalCuts;
```

With Algorithm 13 we are able to find cuts using minimum vertex separators as described in Section 4.4.2. Some parts of the algorithm need explanatory remarks:

- To find a vertex separator in G, we use a *VertexSeparatorGenerator*. That is an object of the class *MinSourceSinkCut* of the library JGraphT[Nav16] applied to an auxiliary graph as described in Section 2.2.3, *Application: Minimum-Weight Vertex Separators*.
- The boolean variable *hasAdditionalCuts* guarantees a sufficiently neat solution of the separation problem. The search for vertex separators (and consequently additional constraints) continues until a cut is added to the current LP relaxation, but at least as long as the value of the x-variable of the center is greater or equal $\frac{1}{|V|}$. *hasAdditionalCuts* is the output of Algorithm 13 and this output is an argument of Algorithm 14. In Algorithm 14, this boolean variable is used for the same purpose.
- The methods *getValue()* and *getValues()* return values of the solution of the LP relaxation to the currently examined branch-and-bound node; *getValue()* is applied to one variable and returns its value whereas *getValues()* is applied to an array of variables concerning a center c (i.e., $x[c]$) and, thus, actually returns a set of values $\left\{x_{c,v} \in [0;1] \middle| \, v \in V\right\}$.
- Given $G = (V, E)$ and a subset $U \subseteq V$, we define $E\big|_U$ as the set of edges restricted to the set of vertices U, that is

$$E\big|_U \coloneqq \left\{ \{u, v\} \in E \middle| \, u, v \in U \right\} = E \cap 2^U.$$

  Consequently, $(U, E\big|_U)$ describes the subgraph of G induced by the vertex set U. In order to find the connected component containing c in this subgraph, we use an object of the class *ConnectivityInspector* which also is part of the library JGraphT.

Algorithm 14: addConnectedComponentsCuts

Data: Adjacency graph $G = (V, E)$, V sorted such that $v \geq w \Leftrightarrow x_{v,v} \geq x_{w,w}$ for $v, w \in V$, Variable[][] x, *hasAdditionalCuts*

Result: Cuts added based on connected components (see Section 4.4.2)

```
1  c ← V.first();
2  while x[c][c].getValue() ≥ 1/V.size() ∨ ¬hasAdditionalCuts do
3      V' ← { v ∈ V | x[c][v] ≥ 1/V.size() };
4      G' ← (V', E|_V');
5      foreach ConnectedComponent U in G' with c ∉ U do
6          B_U ← { w ∈ V \ U | ∃_{u∈U} : {u, w} ∈ E };
7          foreach i ∈ C do
8              if Σ_{j∈B_U} x[c][j].getValue() < x[c][i].getValue() then
9                  addCut("Σ_{j∈B_U} x[c][j] ≥ x[c][i]");
10                 hasAdditionalCuts ← true;
```

Algorithm 14 allows us to add cuts using connected components of the adjacency graph as explained in Sect. 4.4.2. All the methods and sets (except $B_U \subseteq V$) described here are known from Algorithm 13. A further description of B_U, the set of vertices in V adjacent to $U \subseteq V$, can be found in Sect. 4.4.2.

5 Analyzing the supply and detecting spatial patterns of urban green spaces via optimization

The following chapter is mainly taken from a joint work with Benjamin Niedermann and Jan-Henrik Haunert [ONH19]. The presented article is an extension of a preliminary version (together with Sven Lautenbach) [NOLH18] which finished runner-up in the competition for the best-paper award at the GI Science 2018 conference. We present a new methodology for evaluating the green-space of a city from idea to visualization. Furthermore, we test our algorithm for exemplary data of more than fifty German cities and demonstrate how to read and analyze the results including the resulting visualization. For that purpose, we emulate a flow of residents to urban green spaces (see Section 2.2.3). Based on this flow network, we model a linear program (Section 2.3.1) for assigning residents to green spaces. This assignment is evaluated with the help of a predefined cost function that considers, among other criteria, the distances between green spaces and residential areas. Afterwards, we aggregate residential areas using the same pool of green spaces and, vice versa, we aggregate green spaces that are visited from the same residential areas. This way, we detect patterns in the accessibility of green space in an urban area.

Abstract
Green spaces in urban areas offer great possibilities of recreation, provided that they are easily accessible. Therefore, an ideal city should offer large green spaces close to where its residents live. Although there are several measures for the assessment of urban green spaces, the existing measures usually focus either on the total size of all green spaces or on their accessibility. Hence, in this paper, we present a new methodology for assessing green-space provision and accessibility in an integrated way. The core of our methodology is an algorithm based on linear programming that computes an optimal assignment between residential areas and green spaces. In a basic setting, it assigns green spaces of a prescribed size exclusively to each resident such that an objective function that, in particular, considers the average distance between residents and assigned green spaces is optimized. We contribute a detailed presentation on how to engineer an assignment-based method such that it yields plausible results (e.g., by considering distances in the road network) and becomes efficient enough for the analysis of large metropolitan areas (e.g., we were able to process an instance of Berlin with about 130 000 polygons representing green spaces, 18 000

polygons representing residential areas, and 6 million road segments). Furthermore, we show that the optimal assignments resulting from our method enable a subsequent analysis that reveals both interesting global properties of a city as well as spatial patterns. For example, our method allows us to identify neighborhoods with a shortage of green spaces, which will help spatial planners in their decision making.

5.1 Introduction

Urban green spaces affect the quality of life in a variety of manners. In different fields, researchers stressed the significance of green spaces to cities considering socio-cultural [TWKS98, TMS07], health [BRMC05, CMS+07, LM10, WBN14], ecological [PCC+11, SAM06], or economic aspects [KYN07, PV13]. Tyrväinen et al. [TPSdV05] give a detailed overview of the benefits of urban forests and trees, which contribute to urban green-space supply. Many of these benefits also apply for green spaces in general, e.g., the impact on physical and mental health or biotopes for flora and fauna in urban environments. Depending on the focus of research, green spaces comprise, for example, urban parks, playgrounds, forests, or natural areas. Some authors include private urban green spaces as they matter, for example, for environmental subjects [WBN14]. In this work, we consider the recreational value of visiting public urban green spaces [Har04]. Due to their important role there is an increasing interest in measuring and assessing the green-space supply of an urban area [FG09, GRL+16, WBN14].

In order to assess the supply of a city with green spaces quantitatively, spatial planners have developed and suggested various indicators. In particular, indicators for *green-space accessibility* and *green-space provision* have been described [GRM+17]. In this paper, we use these terms as follows:

Definition 5.1. Green-space provision *describes a measure quantifying the total potential of a green space or a set of multiple green spaces with respect to its recreational value.*

Definition 5.2. Green-space accessibility *describes a measure quantifying the effort needed to visit a green space depending on the location of residence of a person. Little effort, i.e. a low value, corresponds to a good accessibility.*

Concerning green-space provision, the area of all available green spaces, for example, is considered and compared with the population size or the total area of the study region [GRM+17, HSS+12]. Various authors agree that the accessibility of a green space is crucial [GCD02, GRM+17, HK15, LM10]. In order to quantify accessibility, one approach is to demand a green space of a certain size within a certain distance from a residential area [CBG08, VHW03]. This criterion does not take the population density of the neighboring residential area into account. Obviously, the green-space supply worsens the more people live in the area [GRM+17, HSS+12]. In this article, we introduce a new methodology to analyze green-space accessibility and green-space provision in an integrated way ensuring that neither small and overcrowded inner-city parks nor vast but hardly accessible green spaces contribute to the assessment disproportionately. In our analyses, we particularly focus on measuring accessibility based on distances in a road network. Furthermore,

we present how our approach can be used to detect patterns in the distribution of green spaces in a city.

To consider green-space provision and accessibility in an integrated way, we assume that every green space can supply only a limited number of residents. This number is referred to as the green space's capacity.

Definition 5.3. *The* capacity *of a green space is the total number of residents it can supply.*

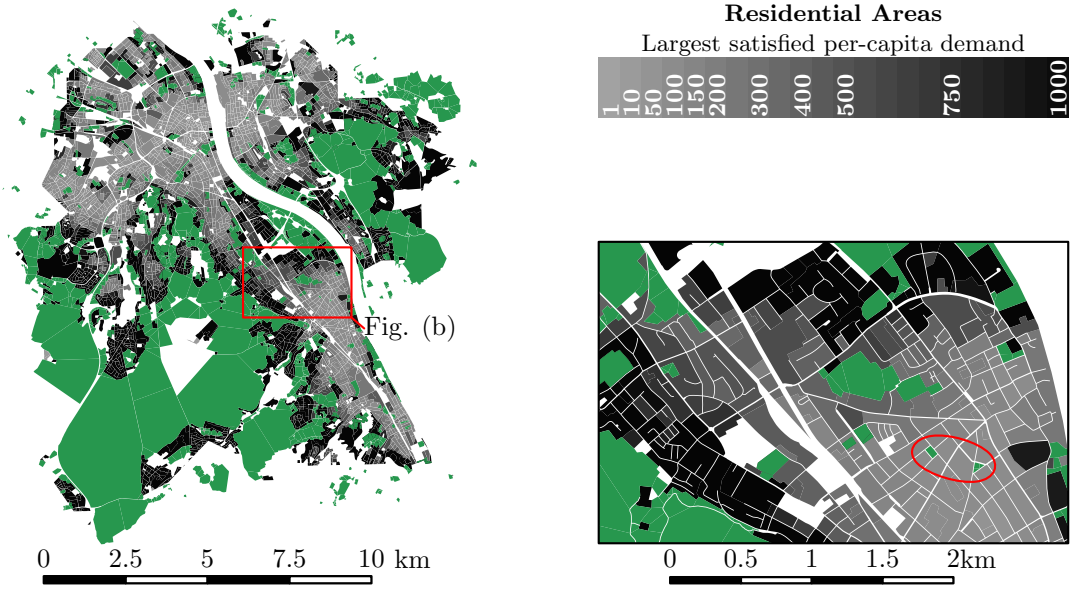
We do not make any restricting assumption concerning the capacities of green spaces, meaning that our method can deal with arbitrary capacities provided as input. For example, one may set the capacity of a green space based on its type (e.g., park or urban forest) and the number of facilities within (e.g, park benches and playgrounds). However, as a simple model we suggest setting the capacity of a green space proportional to its size, following existing recommendations [Sch17]. The factor of proportionality is referred to as per-capita demand.

Definition 5.4. *The* per-capita demand *is the area of green spaces demanded by a single resident.*

We assume that if for a resident the accessibility of a green space is too low then visiting that green space has no recreational value and, thus, is out of choice. This is expressed with the resident's scope.

Definition 5.5. *The* scope *is the maximum effort a resident is willing to spend for accessing a green space.*

Basically, we aim at finding an assignment of a city's residents to the green spaces that respects the residents' scopes as well as the capacities of the green spaces and that is optimal according to a mathematical score function. The assignment is modeled as a flow transporting quantities of residents between residential areas and green spaces, where each residential area is a polygon (of approximately the size of a city block) with a population size. While in Section 5.3 we will present a more detailed discussion of the score function that we optimize, we generally assume that the score increases if more residents are supplied or if the same number of residents is supplied with green spaces that are better accessible. This general property allows us to conduct interesting analyses. For example, for a fixed scope and per-capita demand, the average distance between the residents and the assigned green spaces is a global measure characterizing an entire city. Furthermore, to conduct a more local analysis, we can increase the per-capita demand (and, thus, the competition among the residents) and study the effect on an optimal assignment. In particular, if we increase the per-capita beyond a certain value, some residents will not get a share of the green spaces anymore. Consequently, if the supply of the residents of some residential area collapses already for a relatively low per-capita demand, we consider it as an indication that this particular residential area is lacking a sufficiently large provision with accessible green spaces. Visualizing the *largest satisfied per-capita demand* for all residential areas thus indicates where green spaces are missing, see Figure 5.1. In a similar manner, we can study



(a) For a scope of 1 500 m many residential areas are supplied only for small per-capita demands (light gray).

(b) Smaller green spaces (red oval) improve the situation for nearby residential areas only slightly.

Figure 5.1: Excerpt of our results (see Section 5.4) for a scope defined with a distance of 1 500 m in the road network. Residential areas are depicted with gray polygons, green spaces with green ones. The darker the gray, the higher is the largest satisfied per-capita demand for the corresponding residential area.

how the roles of the green spaces change if the per-capita demand increases. Generally, important green spaces are those to which residents are assigned already for small per-capita demands. We will discuss more possibilities of analyzing the outcomes of our method in Section 5.4.

Although our approach offers a variety of settings for specifying the role of certain green spaces or residential areas, we focus on the most basic set up in which all green spaces and residential areas, respectively, are considered equal apart from their size, i.e. their capacity and their population, respectively. With these settings, we apply our model to 53 German cities. For every data set, i.e. for each city, we run a series of experiments with different settings with respect to the per-capita demand and the scope of the residents. In this work, we extend each of these experiments with a cluster analysis in order to detect patterns in the distribution of urban green spaces. These clusters are defined based on connected components within those parts of the road network that are used by the residents in an optimal assignment. The patterns we detect during the evaluation of our experiments range from global pools of green spaces supplying every resident of a city to sets of local green spaces supplying their neighborhood only.

With respect to the global measure of quality, we consider the average distance to assigned green spaces particularly interesting. Nevertheless, we will introduce a more general objec-

tive function that allows us to distinguish different types of green spaces and residential areas of different demands.

To put our general idea to practice, several design decisions have to be made and technical obstacles have to be overcome. For example, the data set has to be selected to include all relevant green spaces. Furthermore, green spaces and residential areas are usually given as sets of isolated polygons with no direct connection to the segments of a road data set and, thus, additional links have to be established. Concerning green-space provision, one should consider not only the size of a green space but also various other aspects that influence its recreational value like the type of green space, its vegetation or its management [HdVA+17, MRTH10, BØTS06]. Moreover, since the polygons representing residential areas and green spaces may be too large and complex to argue about the distances between them accurately, it may be necessary to partition the polygons into smaller units. All of these aspects are considered in our method in the sense that it offers parameters that should be set by domain experts (e.g., spatial planners). We discuss in detail how these parameters are considered in our method. However, we use rather basic methods and parameter settings in our experiments as our focus lies on presenting the methodology of our approach. Our setup exemplifies how our tools can be used – but, finally, the parameter setting depends on the application and is, consequently, up to domain experts.

In algorithmic terms, we adapt the *transportation problem* [Mun57], which has been studied frequently to decide how to ship a commodity from a set of suppliers to a set of consumers [Gas90]. For assessing green spaces, however, it has not been applied yet. The transportation problem can be solved with specialized algorithms [FJF55] or via linear programming (LP) [FJF56b]. We choose the latter since it can be implemented easily with a mathematical solver and since an LP formulation can be extended easily, for example, to incorporate additional constraints [NW88].

The rest of the paper is structured as follows. After discussing related work (Section 5.2), we introduce a generic network flow model that constitutes the core of our methodology (Section 5.3). We further present how to deploy this model overcoming several technical obstacles. Then, we describe how to use this model for the analysis of green spaces (Section 5.4). We finally conclude the paper with a short outlook on future work (Section 5.5).

5.2 Related Work on Qualitative and Quantitative Analyses of Green Spaces

In this section, we present and discuss a selection of existing approaches for the assessment of urban green spaces.

Baycan-Levent et al. [BLVN09] make clear that assessing the green-space supply of a city is a complex problem. They perform an analysis on several criteria considering among others ecological, economic, and social aspects. With their approach, only the green spaces of an urban area themselves are assessed without taking the residential areas into account: The sheer existence of a high-quality green space improves the result of an assessment of a city regardless of whether its residents are able to access it.

Especially for benefits arising from visiting a green space, like recreation, its accessibility is crucial. Literature is growing on this topic, yet, no outstanding approach for measuring accessibility exists on which researchers can agree [WBN14]. Coombes et al. [CJH10] and Potestio et al. [PPP+09], for example, examine the accessibility of green spaces in the study area with the help of a survey. They consider the distances between the dwellings of participants and the respective closest green space. Other approaches focus on the size or the number of green spaces that are found within a certain radius, e.g. 1 mile, from the residents or, conversely, how many residents live within a certain distance of a green space [MGK+17, NNR+06, OJ07, RER+06, Tal13]. While some of these approaches deal with Euclidean distances, others consider the distance in the road network as it improves the accuracy of a model [Nic01].

Comber et al. [CBG08] examine the green-space supply of a city with respect to its residential areas. They perform a *road-network analysis* in order to determine the accessibility of urban green spaces. With respect to the road network, they consider the percentage of citizens living within a certain radius of green spaces exceeding a minimum size. Whether the available green spaces are sufficient for the number of residents is examined on a global scale only. Furthermore, Comber et al. detect for residential areas whether a green space of adequate size is within a certain distance d or not. If not, no further differentiation takes place: For their assessment methodology, it does not matter whether residents have to walk slightly more than d to the next green space or several times the distance. In order to handle this problem, Comber et al. repeat their analysis for various settings concerning the distance to and the minimum size of the considered green spaces.

Sister et al. [SWW10] use a road-network analysis in order to examine *park pressure*, the ratio of the number of people assigned to a park to its area. This assignment is based on a Voronoi tessellation and, thus, for every resident, only the closest park is considered. Hence, there can be enormous pressure on a park although there are many parks with low pressure nearby but a little bit more distant to the residents. This is a reason why we argue that park pressure is not an ideal tool for assessing the green-space supply. Consequently, we present an approach that takes both distances and capacities into account.

In a recent work, Grunewald et al. [GRM+17] suggested indicators considering both green-space accessibility and provision. For accessibility, they compute the share of inhabitants living within a certain distance from green spaces. Concerning provision, they examine the green-space area per capita both globally and in walking distance from residential areas. A city with green spaces that are accessible for many residents but are of insufficient capacity, e.g. in high-density residential areas, is assessed positively with respect to accessibility; a city with large green spaces accessible only for few, e.g. on the outskirts, gets a good assessment concerning provision. A combination of both leads to a good overall assessment although most of the city's inhabitants have either access to overcrowded green spaces only or limited green-space access. The problem is that Grunewald et al. rather accumulate than combine accessibility and provision criteria. In this paper, we consider green-space provision and accessibility in an integrated manner.

5.3 Methodology

In this section, we present the methodology of our approach. We begin in Section 5.3.1 describing our tool for the analysis of the green-space supply – from the original idea to the implementation capable of processing instances of metropolitan dimensions. In Section 5.3.2, we enhance our approach in order to search patterns within the structure of a computed solution. Subsequently, we describe in Section 5.3.3 in detail how to apply our approach in practice. In Section 5.3.4, finally, we present a small example to which we apply our method.

In the following, we describe a tool set that works on various scales. To begin with, we present the requirements for setting up our model. Generally, the tools can be applied to arbitrarily sized residential areas and green spaces. However, results are more meaningful if the following aspects are considered:

- Green spaces and residential areas should be digitized with such a granularity that a single number suffices for representing the distance between a residential area and a green space.
- If information on the recreational value of green spaces is available, the green spaces should be digitized such that within each green space the recreational value does not vary too much.
- If information on the mobility of residents is available, the residential areas should be digitized such that within each residential area the mobility of the residents does not vary too much. Residential areas with mixed categories of residents (e.g., families and elderly people) could be split into multiple entities, one for each group.

Figure 5.2 depicts two scenes that differ only with respect to the granularity of the green spaces. In Figure 5.2(a), the shortest distance between the residential area r and the green space g is very small although most residents of r would have to walk a rather long way to access the capacious main body of g. Therefore, we argue that the granularity of the green space is not well chosen according to the first requirement. Instead, we suggest partitioning the green space into smaller and more compact polygons as presented in Figure 5.2(b). With this we can distinguish between assignments to near parts of g and assignments to distant parts of g and rate their accessibility differently for the residents in r.

5.3.1 Basic Model

At first, we present a basic model for analyzing the provision and the accessibility of green spaces in combination – the foundation of our further analysis.

Basic Ideas Our model originates from realizing that established tools for assessing urban green spaces focus either on the provision or the accessibility of green spaces. Among the simplest approaches, for example, is analyzing the green-space area that is available per person. Without considering the accessibility simultaneously, vast green spaces on the outskirts of a city have a higher influence on the assessment of a city than green spaces in the inner city to which residents have access to in their daily life. Conversely, considering accessibility



Figure 5.2: Green spaces (green) and residential areas (red) with different granularities of digitization.

criteria like, for example, mean distance only leads to positive assessments caused by small inner city parks of insufficient size.

Our model interweaves both measures and overcomes their shortcomings. It rests on data of residential areas with population information, green spaces with capacities, and a network connecting in between. With our approach, we seek an assignment of shares of green spaces to residents maximizing a score function. The score respects the green spaces' capacities and the residents' scope, i.e. the maximum distance residents are willing to cover, while considering green-space preferences of the residents. Our model is adaptable to a variety of requirements. Altering the capacity of certain green spaces allows the user to differentiate between different kinds of green spaces. Altering the scope of the residents of certain residential areas allows the user to consider, for example, the residents' mobility. Finally, the user can adapt the underlying network to different means of transportation.

Formalization Let (R, G) be an urban area composed of a set R of residential areas and a set G of green spaces. For each residential area $r \in R$ a number $I(r)$ of residents is defined. For every green space $g \in G$ the *capacity* $C(g)$ yields information about the number of residents that can be served by g. Then, we interpret a triplet $(r, g, i) \in R \times G \times \mathbb{R}^+$ as the number i of residents of r who are assigned to green space g. An *assignment* $A \subseteq R \times G \times \mathbb{R}^+$, i.e., a set of such triplets respecting both the capacity of g and the population size of r, is sought. That is, $\sum_{(r,\cdot,i)\in A} i \leq I(r)$ holds for every $r \in R$ and $\sum_{(\cdot,g,i)\in A} i \leq C(g)$ holds for every $g \in G$. In order to avoid ambiguity, any assignment A is assumed to contain no triplet (r, g, i) with $i = 0$. Various assignments exist but they differ in quality. Hence, a *score* function $s\colon R \times G \to [0, 1]$ assessing the combination of a residential area and a green space is introduced that can be defined by the user of our tool set. High scores for $r \in R$ and $g \in G$ correspond to a high preference of g by the residents of r. As long as this score function is precomputable it can be chosen arbitrarily. Getting back to maximizing the score, this objective is modeled as maximizing $\sum_{(r,g,i)\in A} s(r, g) \cdot i$, the *total score*. We refer to this problem as GreenSpaceAssignment.

From a computational point of view, GreenSpaceAssignment is a maximum flow problem in a complete bipartite graph formed by R and G; compare Figure 5.3(a). This flow

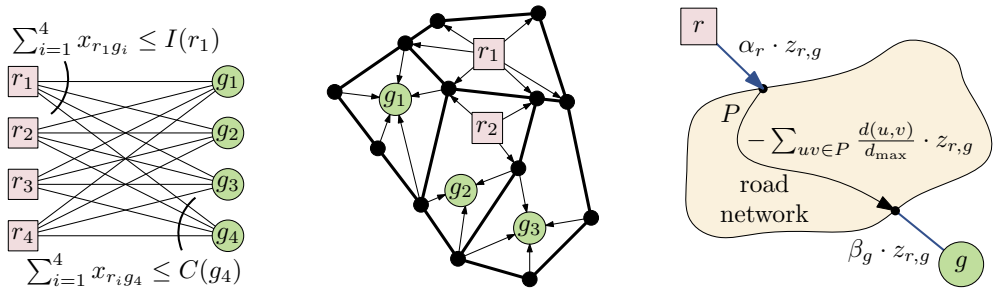


Figure 5.3: Assignment Model with residential areas (red boxes) and green spaces (green circles). (a) Illustration of a generic assignment model. (b) Service network $N = (V \cup R \cup G, E \cup F)$ based on the road network $H = (V, E)$ (black vertices, fat edges), residential areas R and green spaces G. (c) Flow $z_{r,g}$ is transmitted from the residential area r to the green space g through the road network on the shortest path P. The flow creates the value given in Equation (5.10). As only the flow from r to g is considered, for every edge $uv \in P$ the inequality $z_{r,g} \leq x_{uv}$ holds.

problem can be subsumed into the following linear program in which the variable $x_{r,g}$, defined for every pair $(r, g) \in R \times G$, expresses the number of residents of r who are assigned to g.

$$\max \quad \sum_{r \in R} \sum_{g \in G} s(r,g) \cdot x_{r,g} \tag{5.1}$$

$$\text{such that} \quad \sum_{g \in G} x_{r,g} \leq I(r) \qquad \text{for every } r \in R \tag{5.2}$$

$$\sum_{r \in R} x_{r,g} \leq C(g) \qquad \text{for every } g \in G \tag{5.3}$$

Distance-based Score Function Assuming that residents prefer nearby and – apart from size or proximity – attractive green spaces [HdVA+17, SWW10], we introduce the following specialization of the presented model. As for matters of accessibility, we suggest to define the score dependent on a distance function d for residential areas and green spaces. The choice of the distance function is up to the user of our tool set. In the next paragraph, our advanced model is presented and here, the distance is defined as the walking distance between residential areas and green spaces. Depending on the application, one may consider walking distances only, include public transportation, or design the distance, for example, based on car-travel distance. However, according to the distance d, a *scope* $d_{\max}$ is introduced, the maximum distance any resident is willing to cover to reach an arbitrary green space. Considering this scope, we define a basic score function

$$s(r,g) = 1 - \frac{d(r,g)}{d_{\max}}. \tag{5.4}$$

For a pair (r, g), maximum score close to 1 is achieved if r and g are as close as possible. If r and g are $d_{\max}$ apart, no score is gained by assigning residents from r to g. For distances

$d(r,g)$ larger than $d_{\max}$ setting $x_{r,g} = 1$ contributes a negative score to the objective function and, thus, will not occur in an optimal solution. Figure 5.4 (blue line) visualizes how this score function depends on the distance between the residential area r and the green space g as well as the scope $d_{\max}$.

However, we suggest to include information about the green spaces beyond size, capacity and location. Thus, this model can be extended to

$$s(r,g) = \alpha_r + \beta_g - \frac{d(r,g)}{d_{\max}} \tag{5.5}$$

where the values α_r and β_g represent additional influence of a specific residential area r or a specific green space g on the score under consideration, see the orange line in Fig. 5.4. Higher values for α_r and β_g counteract the negative effect of $\frac{d(r,g)}{d_{\max}}$ on the score. For a residential area r, this can be interpreted for example as a higher mobility of the residents of r. For a green space g, it can be used for expressing an increased attractiveness of g compared to other green spaces. This influences the distance residents are willing to cover [GCD02]. We call the result *effective scope*. Depending on the combination of green space g and residential area r, the effective scope is $d_{\max} \cdot (\alpha_r + \beta_g)$. For s to map to $[0,1]$ it is necessary that $\alpha_r + \beta_g \leq 1$ for any combination of a residential area r and a green space g. This way, the highest effective scope, i.e. for $\alpha_r + \beta_g = 1$, expresses the distance residents with the highest mobility are willing to cover to reach the most attractive green space. Thus, in this case, effective scope and scope are the same.

Network-based Assignment Model The LP formulation consisting of equations (5.1)–(5.3) solves GreenSpaceAssignment but is applicable only for cities of moderate size. The quadratic number of variables in combination with the desired level of detail of the analysis asks too much even of modern server systems when it comes to larger cities. Therefore, we introduce a road-network based formulation using a number of variables that is linear in the size of the number of green spaces, residential areas and the size of the road network. This way, the analysis of metropolitan cities is possible.

Based on a road network given as a directed geometric graph $H = (V, E)$, a *service network* $N = (V \cup R \cup G, E \cup F)$ is built. Its vertex sets consists of V, the vertex set of H, as well

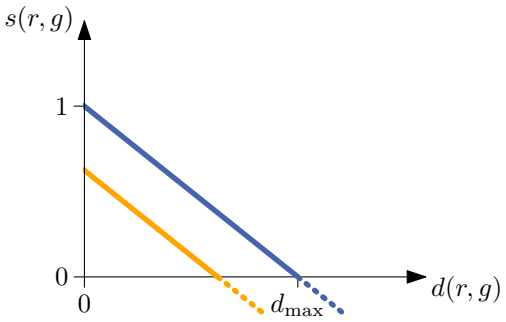


Figure 5.4: Score functions for $\alpha_r + \beta_g = 1$ (blue, see in particular Eq. 5.4) and $\alpha_r + \beta_g < 1$ (orange, see Eq. 5.5) plotted versus the length of the shortest path.

as one vertex for every residential area $r \in R$ and every green space $g \in G$. These vertices are linked to the edges E of the road network with additional edges F completing the edge set of N to $E \cup F$, see Figure 5.3(b). In detail, every residential area is connected to access points within the road network via outgoing edges. Likewise, every green space is connected to access points via incoming edges. These access points connect residential areas and green spaces with the road network. We use a generic approach to define access points: A vertex in the road network serves as an access point to a green space if it is contained in a buffer of, for example, 100 m around the green space. We use the same method to define access points for residential areas. However, the precision of the model can be increased by using real-world access point like park or building entrances. Although, using the latter as access points makes only sense if population data is available for every building.

The weight d of a service-network edge is defined as follows. For edges $(r, v) \in F$ connecting a residential area r with an access point v, the length $d\,(r, v) = \alpha_r$ is defined. Likewise, for edges $(u, g) \in F$ connecting an access point u with a green space g, the length is defined as $d\,(u, g) = \beta_g$. For every edge e that stems from the road network, i.e. $e \in E$, the edge weight is inherited from the road network. Depending on the underlying road network this may be, for example, the geodesic length of the edge or the time it takes to travel along it.

With the service network being set up, an LP formulation can be introduced that is less complex than the previous one in the sense that a smaller number of variables is used. Here, for every edge $e \in E \cup F$, a variable x_e is used to model the flow on e. This flow represents the number of residents using edge e. We use these variables to formulate the LP as follows.

$$\max \sum_{r \in R} \sum_{(r,v) \in F} \alpha_r \cdot x_{(r,v)} + \sum_{g \in G} \sum_{(u,g) \in F} \beta_g \cdot x_{(u,g)} - \sum_{(u,v) \in E} \frac{d\,(u, v)}{d_{\max}} x_{(u,v)} \tag{5.6}$$

such that

$$\sum_{(r,v) \in F} x_{(r,v)} \leq I(r) \qquad \text{for every } r \in R \tag{5.7}$$

$$\sum_{(u,g) \in F} x_{(u,g)} \leq C(g) \qquad \text{for every } g \in G \tag{5.8}$$

$$\sum_{(u,v) \in E \cup F} x_{(u,v)} = \sum_{(v,w) \in E \cup F} x_{(v,w)} \qquad \text{for every } v \in V \tag{5.9}$$

Constraint (5.7) bounds the outflow of a residential area r with its population size $I(r)$. Likewise, Constraint (5.8) respects a green space g's capacity $C(g)$. Constraint (5.9) demands for every road-network vertex that its inflow equals its outflow and, thus, preserves the flow within the road network. Altogether, these constraints allow flow to emerge at residential areas only and to drain away at green spaces only. The flow produced at a residential area does not exceed the respective population size and a sink at a green space has to respect the corresponding capacity.

The objective function is set up to correspond to the objective of GreenSpaceAssignment. This becomes clearer when the contribution of a flow $z_{r,g}$ from a specific residential

area r to a specific green space g is considered. The flow $z_{r,g}$ contributes $\alpha_r \cdot z_{r,g}$ to the objective by leaving the source r. Along its path P through the network, it contributes $-\frac{d(u,v)}{d_{\max}} \cdot z_{r,g}$ for every network edge uv used, i.e. $-\sum_{uv \in P} \frac{d(u,v)}{d_{\max}} \cdot z_{r,g}$ in total. Furthermore, by draining away at green space g, the flow contributes $\beta_g \cdot z_{r,g}$. Since there is no limitation to the usage of a network edge and since the LP is a maximization problem, the flow uses a shortest path from r to g, i.e. P has length $d(r,g)$, see Figure 5.3(c). This means, that $z_{r,g}$ contributes

$$\left(\alpha_r + \beta_g - \sum_{(u,v) \in P} \frac{d(u,v)}{d_{\max}}\right) \cdot z_{r,g} = s(r,g) \cdot z_{r,g} \tag{5.10}$$

to the objective. Considering the network flow in total, the objective corresponds to $\sum_{r \in R} \sum_{g \in G} s(r,g) \cdot z_{r,g}$, the objective of GreenSpaceAssignment.

Although, with this objective function, residents prefer nearby green spaces, results differ from solutions obtained with simple models. Consider, for example, a greedy approach assigning residents to the closest green space with remaining capacities. Figure 5.5(a) depicts an example with residential areas with population $I(r) = 1$, green spaces with capacity $C(g) = 1$, and road edge length a, b, and c with $b < \min\{a, c\}$ with different feasible assignments. In this case, a greedy algorithm finds a solution assigning the residents of r_1 to green space g_2 if $b \leq d_{\max}$ (orange in Figure 5.5(b)) and, additionally, the residents of r_2 to g_1 if $a + b + c \leq d_{\max}$ (blue in Figure 5.5(b)). With our approach, a solution with assignments to both g_1 and g_2 (red in Figure 5.5(c)) are possible even for scopes smaller than $a + b + c$. This means that residents from residential area r_1 are willing to make room for residents from r_2 within certain bounds. The red assignment achieves the maximum score for $d_{\max} \geq \frac{a+c-b}{(\alpha_{r_2} + \beta_{g_1})}$ among all assignments presented in Figure 5.5(b) and (c).

5.3.2 Analyzing the Resulting Clusters

For a deeper analysis of urban green spaces, we continue with an analysis of supply clusters based on connected components defined by the resulting flow in the service network.

The result of our tool presented in Section 5.3.1 is an assignment of the variables $x_{(u,v)}$ for every $(u,v) \in E \cup F$. A variable $x_{(u,v)}$ indicates how many residents use the corresponding edge (u,v) in an optimal solution to network-based GreenSpaceAssignment. Based on this information, the edges $E \cup F$ of the service network can be divided into two groups: edges that play a role in an optimal solution and those which do not. This categorization allows us to detect clusters of used edges within the network by applying, for example, a depth-first search on the service network [CLRS09]. This way, further structures of the green-space supply become visible.

The clusters are defined based on the service network weighted with the solution to GreenSpaceAssignment. More precisely, we consider the graph

$$N' = (V \cup R \cup G, \{e \in E \cup F \mid w(e) > 0\}),$$

i.e. only edges $e \in E \cup F$ which are part of the computed solution. In N' we look for connected components. Searching these connected components, we ignore edge directions,

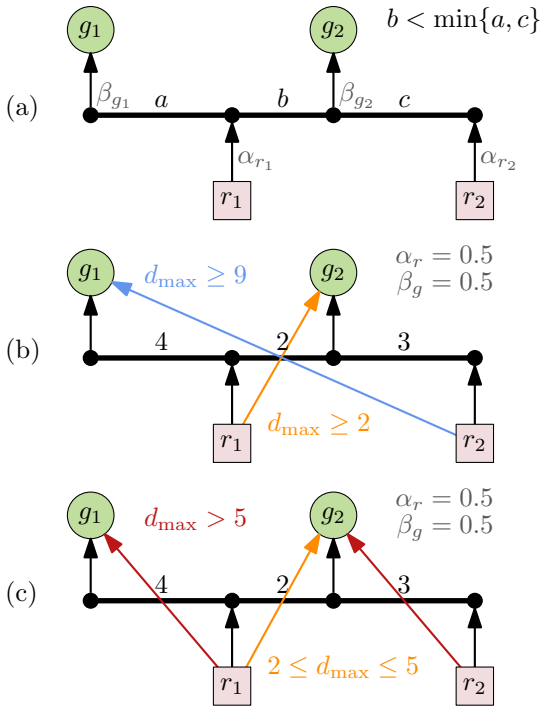


Figure 5.5: Small example for visualizing differences to greedy approaches for assigning residents to green spaces. To keep things simple, every residential area r is inhabited by one resident and every green space g's supply suffices for one resident only. (a) Set up with $b < \min\{a, c\}$. (b) Example with $(a, b, c) = (4, 2, 3)$. Optimal solutions for different scopes for the greedy approach. For $d_{\max} \geq 2$, the orange assignment takes place. For $d_{\max} \geq 9$, the blue assignment is active in addition. (c) Example with $(a, b, c) = (4, 2, 3)$. Optimal solutions for different scopes for our approach. The orange assignment takes place for $2 \leq d_{\max} \leq 5$. For $d_{\max} = 5$, the red assignment has the same objective value, for larger scopes it is superior.

i.e. all outgoing edges of a single residential area are part of the same cluster and, likewise, all incoming edges of a single green space. In the following, we refer to these connected components as *clusters*.

5.3.3 Deployment

In this section, we state in detail how we deploy our network-based model. As we want to present our methodology, we use the model in its most general way, i.e., without emphasizing any particular residential area or green space, for example, by specifying certain α_r or β_g values. Furthermore, we explain how to use the results obtained to get more information about the quality of a city's green-space supply as described in Section 5.3.2. For this purpose, the residential areas R and the green spaces G need to be given as simple polygons. For each residential area also the number of residents needs to be provided. Furthermore, our algorithm takes the road network as a graph $H = (V, E)$ with geometric embedding as

an input. In the first phase of deployment, the given data is preprocessed in order to obtain an instance of GreenSpaceAssignment which gets solved in the second phase. In a third phase, a detailed analysis is applied.

First Phase – Preprocessing

1. *Partitioning the green-space polygons*
   As green-space polygons may be too large to express the distance between them and some residential area accurately with a single value, we suggest partitioning these polygons into smaller units. The user should aim for a granularity comparable to the granularity of the residential areas. Of course, it is possible to partition residential areas as well to achieve a finer overall granularity but this makes sense only if information about the distribution of the residents is available. For the partitioning of the green spaces, we apply the algorithm of Haunert and Meulemans [HM16]. It decomposes a simple polygon into a minimum number of simple polygons such that each of the resulting polygons is sufficiently compact with respect to a measure of dilation from graph theory. The resulting set of green spaces represented by compact polygons constitute the set G in the following.
2. *Determining/Enriching the road network with access points to the polygons*
   Within the road network, access points for the residential areas R and the green spaces G need to be selected or, in case of available more precise data, created. For this purpose, we buffer the polygons; we use an offset of 100 m. Road vertices that are contained in the buffered polygons serve as access points. Thus, green spaces can be accessed by roads passing nearby. Likewise, residents can enter the road network via close roads.
   This step can be skipped if access points for residential areas and green spaces are given in the input data.
3. *Building the service network*
   In order to set up the service network, it is necessary to connect the green spaces G and the residential areas R to the road network. For a residential area $r \in R$, we suggest to simply create an edge rv for every access point v of the corresponding polygon. Similarly, we introduce an edge ug for the access points u to a green space $g \in G$. Denoting the set of these additional edges by F, the service network $N = (V \cup R \cup G, E \cup F)$ results.
4. *Reducing the complexity of the service network*
   Since degree-2 vertices of the service network do not have an influence on the result we eliminate them in order to improve the algorithm's running time. This is done by recursively replacing the incident edges of a degree-2 vertex with an edge connecting its neighbors. The new edge's length is derived from the lengths of both original ones.
5. *Further reduction of the complexity of the service network*
   Due to the problem formulation of GreenSpaceAssignment, in the service network, only shortest paths between green spaces and residential areas are of interest. Any vertex that is not part of such a shortest path gets removed. For any other vertex, its *accessibility* is annotated. We define the reachability of a vertex as its minimum distance to a residential area. It is used for a speed-up of our algorithm in the second phase.

Second Phase – Linear Programming The first phase provides an almost complete instance of GreenSpaceAssignment. Information about the capacities of green spaces and the scope is still lacking. Hence, we systematically explore a variety of capacities and scopes and apply them to the defined problem. We use an areal green-space size γ that is necessary to satisfy a resident, the *per-capita demand*. A general definition for the capacity, $^{\text{area of the green space}}/_{\gamma}$, follows. In the experiments, we consider a variety of parameters as choosing a single one is difficult. Hence, we consider a set Γ of different per-capita demands. Likewise, a set D of various scopes is considered. In order to solve GreenSpaceAssignment for a pair $(\gamma, d) \in \Gamma \times D$, the corresponding linear program is completed with the following definitions. The capacity of a green space $g \in G$ is set to $^{\text{area}(g)}/_{\gamma}$ and, finally, the scope $d_{\max}$ is set to d. By considering the reachability of a vertex in the service network (see First Phase, Step 5) it is possible to decide beforehand whether the vertex may play a role in the solution to the linear program. Network edges to vertices with a distance larger than the scope do not need to be considered at all which allows us to set up smaller linear programs.

For every pair $(\gamma, d) \in \Gamma \times D$, the solution obtained provides for every edge in $E \cup F$ information about the number of residents using that edge.

Third Phase – Detailed Analysis The solution of an instance of GreenSpaceAssignment yields much more than simply the overall score of a city. Applying the solution of one GreenSpaceAssignment instance to the graph of service network one obtains a weighted graph with an edge weight $w\colon E \cup F \to \mathbb{R}$ indicating the number of residents using the considered edge.

1. *Analyzing edges connecting green spaces and residential areas to the network*

   Analyzing the edges of F, i.e. edges connecting green spaces and residential areas to the road network, yields information about the situation for the respective green space or residential area. For every green space $g \in G$, the term $\sum_{(u,g)\in F} w(u, g)$ yields the number of residents that are supplied by g. Conversely, for every residential area $r \in R$, the term $\sum_{(r,v)\in F} w(r, v)$ tells how many residents of r have access to the green spaces such that their per-capita demand is fulfilled.

   For a fixed scope, we consider the following measures:

   - For each $r \in R$ its *largest satisfied per-capita demand*: the largest per-capita demand $\gamma \in \Gamma$ for which all residents of r are assigned to green spaces.
   - For each $g \in G$ its *smallest relevant per-capita demand*: the smallest per-capita demand $\gamma \in \Gamma$ such that g is used in the assignment.

   Furthermore, for a fixed per-capita demand, we define two measures as follows:

   - For each $\gamma \in \Gamma$ the *smallest scope satisfying all residents*: the smallest scope such that the per-capita demand is satisfied for every resident.
   - For each $\gamma \in \Gamma$ the *average distance to assigned green space*: the average distance between residents and assigned green spaces, assuming an infinite scope. If residents remain unassigned (i.e., there is not enough supply for the respective per-capita demand) the measure is undefined.

2. *General analysis of service network edges*

   The edges of E, i.e. edges of the road network, yield further information, too. In combination with the links F to green spaces and residential areas, for example, it is possible to detect clusters of residential areas that share green spaces. Thus, it is possible to detect whether, in a city, the green-space supply is provided by a global pool of green spaces supplying every resident or by a set of local green spaces supplying their neighborhood only. We suggest to compare the resulting clusters with respect to the total number of residents of the residential areas within. If the polygons were subdivided in the preprocessing phase (step 1 of first phase) we define that every two polygons stemming from the same original polygon belong to the same cluster.

5.3.4 A running example

In Figure 5.6, a scenario is given with three green spaces g_i with $i \in \{1, 2, 3\}$ and four residential areas r_i with $i \in \{1, 2, 3, 4\}$. Furthermore, a road network is given such that the shortest path from the access points of residential areas to those of green spaces can be expressed with five generalized edges (t, u), (u, t), (v, u), (u, w), and (w, u). The service network is completed by edges connecting the residential areas and the green spaces with the corresponding access points. Accordingly, the linear program uses the variables $x_{(r_1,t)}$, $x_{(r_2,t)}$, $x_{(r_2,u)}$, $x_{(r_3,v)}$, and $x_{(r_4,w)}$ for edges connecting residential areas to the road network, $x_{(t,g_1)}$, $x_{(u,g_2)}$, and $x_{(w,g_3)}$ for edges to green spaces, and $x_{(t,u)}$, $x_{(u,t)}$, $x_{(u,v)}$, $x_{(u,w)}$, and $x_{(w,u)}$ for edges of the road network.

Corresponding to Formula 5.6, the objective function is given as

$$\begin{aligned} \max \quad & x_{(r_1,t)} + x_{(r_2,t)} + x_{(r_2,u)} + x_{(r_3,v)} + x_{(r_4,w)} \\ & - \frac{4}{d_{\max}}(x_{(u,t)} + x_{(t,u)}) - \frac{3}{d_{\max}} x_{(v,u)} - \frac{5}{d_{\max}}(x_{(u,w)} + x_{(w,u)}). \end{aligned}$$

Constraints 5.7 considering the population sizes of the residential areas are given as follows:

$$\begin{aligned} x_{(r_1,t)} &\leq 10, & x_{(r_3,v)} &\leq 10, \\ x_{(r_2,t)} + x_{(r_2,u)} &\leq 20, & x_{(r_4,w)} &\leq 20. \end{aligned}$$

Likewise, the capacities of the green spaces are respected, see Formula 5.8. The per-capita demand γ is used to express the capacity of a green space:

$$\begin{aligned} x_{(t,g_1)} &\leq \frac{500\,\mathrm{m}^2}{\gamma}, \\ x_{(u,g_2)} &\leq \frac{500\,\mathrm{m}^2}{\gamma}, \\ x_{(w,g_3)} &\leq \frac{2\,000\,\mathrm{m}^2}{\gamma}. \end{aligned}$$

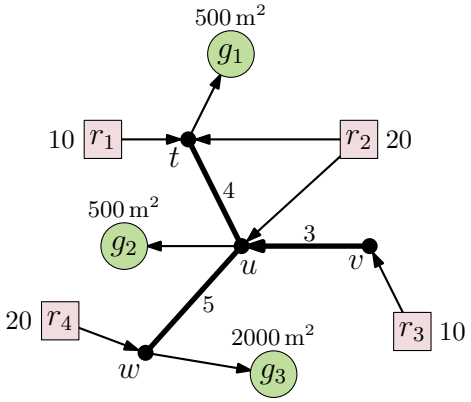


Figure 5.6: Scheme of an exemplary setup. Green spaces g are given with their size. Residential areas r are given with their population. Only edges that are part of a shortest path from a residential area to a green space are given; edges of the road network are presented with their cost, edges related to the access from residential areas and to green spaces, respectively, are provided with $\alpha_r = 1$ and $\beta_g = 0$, presented with thin lines.

Finally, the constraints preserving the flow are defined for the vertices of the road network, see Formula 5.9:

$$\begin{aligned} x_{(r_1,t)} + x_{(r_2,t)} + x_{(u,t)} &= x_{(t,g_1)} + x_{(t,u)}, \\ x_{(r_2,u)} + x_{(t,u)} + x_{(v,u)} + x_{(w,u)} &= x_{(u,g_2)} + x_{(u,t)} + x_{(u,w)}, \\ x_{(r_3,v)} &= x_{(v,u)}, \\ x_{(r_4,w)} + x_{(u,w)} &= x_{(w,g_3)} + x_{(w,u)}. \end{aligned}$$

In Table 5.1, exemplary solutions can be found that are optimal with respect to different parameter settings. In the first example (see Figure 5.7(a)), i.e. the first column, although the per-capita demand is set to only $1\,\mathrm{m}^2$ not all residents are satisfied. As the scope is set to 2.5, no green space is accessible for residents of r_3. In the second example (Fig. 5.7(b)), the scope is extended to 10. Here, all residents have access to the green spaces and their demand is fulfilled. The third example (Fig. 5.7(c)) with $\gamma = 50\,\mathrm{m}^2$ and $d_{\max} = 7$ is an example where sufficient green spaces are available and actually accessible. Nevertheless, the residents of r_3 remain unsatisfied since making room in green space g_2, i.e. assigning all residents of r_2 to g_3, is, with respect to the objective function, not worth the extra expenditure. The forth example (Fig. 5.7(d)) is set up with the same per-capita demand $\gamma = 50\,\mathrm{m}^2$ and a larger scope $d_{\max} = 10$. Here, 20 out of 30 residents of r_2 and r_3 are assigned to g_3. Due to the used network-flow approach, it is not possible to give more precise information such as the residents of which residential area are assigned to which green space. In the final example, the per-capita demand is too high to supply every resident with their share of the green spaces. Two optimal solutions are presented. In the first solution (Fig. 5.7(e)), g_1 supplies r_1 and r_1 uses g_1 only; the same holds for r_2 and g_2 as well as r_4 and g_3. Thus, we detect three clusters (gray in Fig. 5.7). In the second solution (Fig. 5.7(f)), g_1 supplies both r_1 and r_2. Consequently, there exist only two clusters (the other one being formed by r_4 and g_3). Such

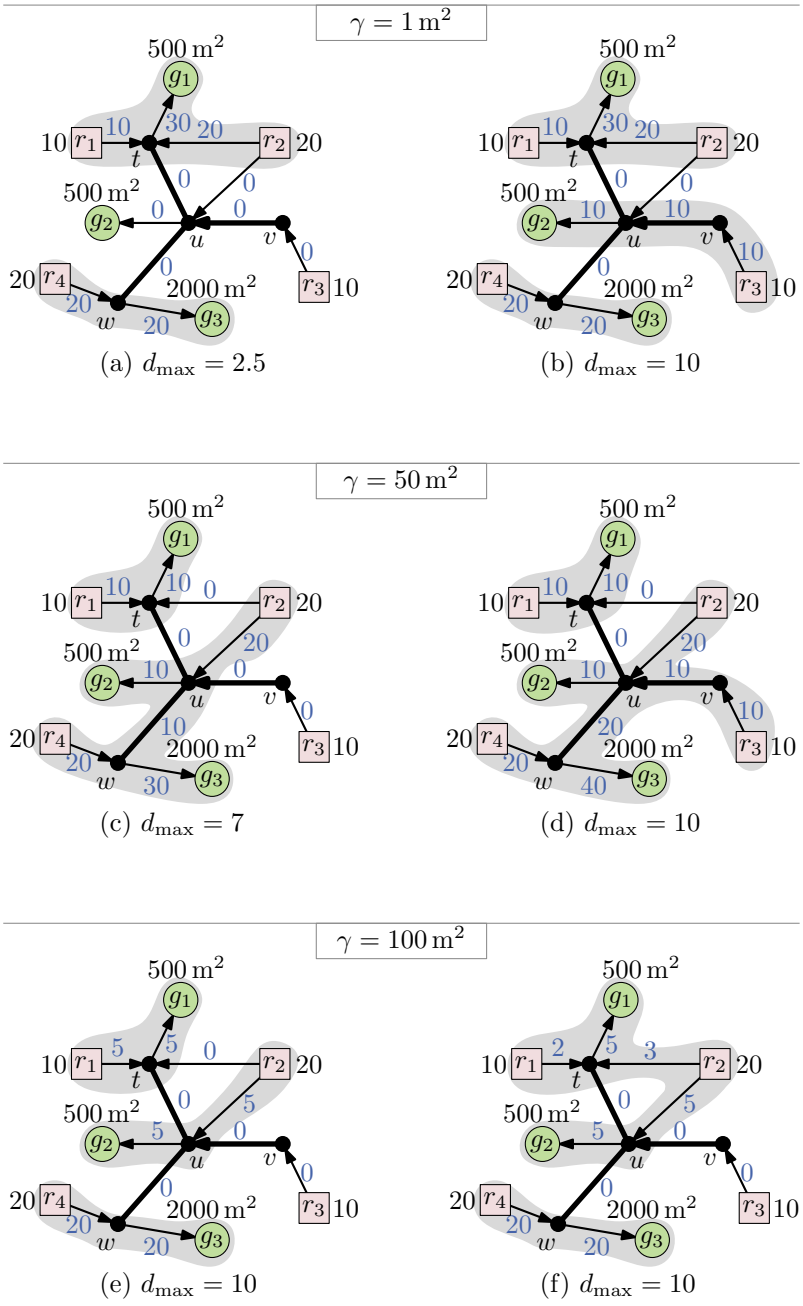


Figure 5.7: Visualization of results for the example presented in Figure 5.6. Each edge's flow is annotated in blue, see variable assignment in the corresponding columns of Table 5.1. The figures depict optimal results for specific per-capita demands γ and scopes $d_{\max}$. Gray regions visualize connected components detected in the cluster analysis.

Table 5.1: Variable assignment maximizing the objective function, see Formula 5.6, for different per-capita demands γ and scopes $d_{\max}$.

γ	$1\,\mathrm{m}^2$	$1\,\mathrm{m}^2$	$50\,\mathrm{m}^2$	$50\,\mathrm{m}^2$	$100\,\mathrm{m}^2$	
$d_{\max}$	2.5	10	7	10	10	
Fig. 5.7	(a)	(b)	(c)	(d)	(e)	(f)
$x_{(r_1,t)}$	10	10	10	10	5	2
$x_{(r_2,t)}$	20	20	0	0	0	3
$x_{(r_2,u)}$	0	0	20	20	5	5
$x_{(r_3,v)}$	0	10	0	10	0	0
$x_{(r_4,w)}$	20	20	20	20	20	20
$x_{(t,g_1)}$	30	30	10	10	5	5
$x_{(u,g_2)}$	0	10	10	10	5	5
$x_{(w,g_3)}$	20	20	30	40	20	20
$x_{(t,u)}$	0	0	0	0	0	0
$x_{(u,t)}$	0	0	0	0	0	0
$x_{(v,u)}$	0	10	0	0	0	0
$x_{(u,w)}$	0	0	10	20	0	0
$x_{(w,u)}$	0	0	0	0	0	0

ambiguities occur in particular when residential areas and green spaces share access points. Apart from these cases, in real world data, exactly identical shortest path lengths are rather improbable. Nevertheless, this issue stresses the importance of an appropriate selection of access points.

5.4 Experiments and the methodology for the evaluation

In this section, we describe our experimental evaluation. As stated earlier, our focus lies on presenting the usefulness of our methodology for the analysis of urban green spaces rather than the analysis itself. In the following, we demonstrate how to gain and present information from the obtained results.

5.4.1 Data

For the moment, accessible data is hard to find for our experiments. Our experiments are based on population estimation data stemming from the Urban Atlas 2012[1]. The data publisher does not provide the data for download anymore but has announced to publish improved population estimates soon. For presenting our model, however, the data is appropriate. Since the main motivation is not the analysis of a specific city's supply of urban

[1] © European Union, Copernicus Land Monitoring Service 2018, European Environment Agency (EEA). `https://land.copernicus.eu/`

green spaces but the presentation of a set of tools for running such an analysis, we continue with this model data set based on 53 urban areas in Germany. For every city in this data set, we obtain information about the residential areas in the city and about green spaces in the city and its surroundings. The residential areas are extracted as simple polygons and typically represent one housing block. Furthermore, each residential area is annotated with its population size. Likewise, green spaces are extracted for the city and its surroundings. We filter this set by considering only those polygons that are tagged as *forest*, *green urban area*, or *sports and leisure facility*. In our experiments, we examine cities with a population size ranging from 33 thousand to 2.4 million; on average, the cities under consideration have 285 thousand residents. These cities provide between 8.6 km^2 and 6 560 km^2 of green space and 659 km^2 on average. A more detailed overview is given in the first three columns of Figure 5.8.

For each city, the road network is extracted from OpenStreetMap[2]. The extract is of sufficient size to contain every shortest path between an urban area and a green space.

5.4.2 Setup

For the second phase, additional parameters need to be defined. In order to keep things simple, the residential areas and the green spaces, respectively, are considered to be matched evenly. That is, α_r has the same value for every residential area $r \in R$. Likewise, β_g has the same value for every green space $g \in G$. Setting $\alpha_r + \beta_g = 1$ for every pair $(r, g) \in R \times G$, for example with $\alpha_r = 1$ and $\beta_g = 0$, guarantees that the distance each resident is willing to cover is the globally defined scope $d_{\max}$. We set up a series of experiments for a combination of different per-capita demands and scopes. For both parameters, example values and recommendations can be found in the literature [BTA+07, CBG08, FG09, Nat19, Sch17]. We consider a set $\Gamma = \{1, 10\} \cup \{50 \cdot i | 1 \leq i \leq 20\}$ of various per-capita demands in m^2 and a set $D = \{0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 60, \infty\}$ of scopes in km. Then, we solve GreenSpaceAssignment for each pair $(\gamma, d) \in \Gamma \times D$. Additionally, for each $\gamma \in \Gamma$ the smallest scope D_γ for which every resident is satisfied is computed.

In the linear program, the variables formalizing the flow of residents are chosen to be continuous. Compared to integer variables, this improves the running time significantly. In consequence, single residents are not considered as such and may be distributed to more than one green space. This is in accordance with this model being used for analyzing the distribution of green spaces rather than setting up rules for assigning residents.

The experiments were performed on an Intel® Xeon® CPU E5-1620 processor. The machine is clocked at 3.6 GHz and has 32 GB RAM. For the implementation, we used Python and Java. In the first and the third phase we utilized QGIS 2.18.14[3]. For the linear programming in the second phase, we used Gurobi 7.0.2[4].

[2] © OpenStreetMap contributors, `https://www.openstreetmap.org/`
[3] `https://www.qgis.org/`
[4] `https://www.gurobi.com/`

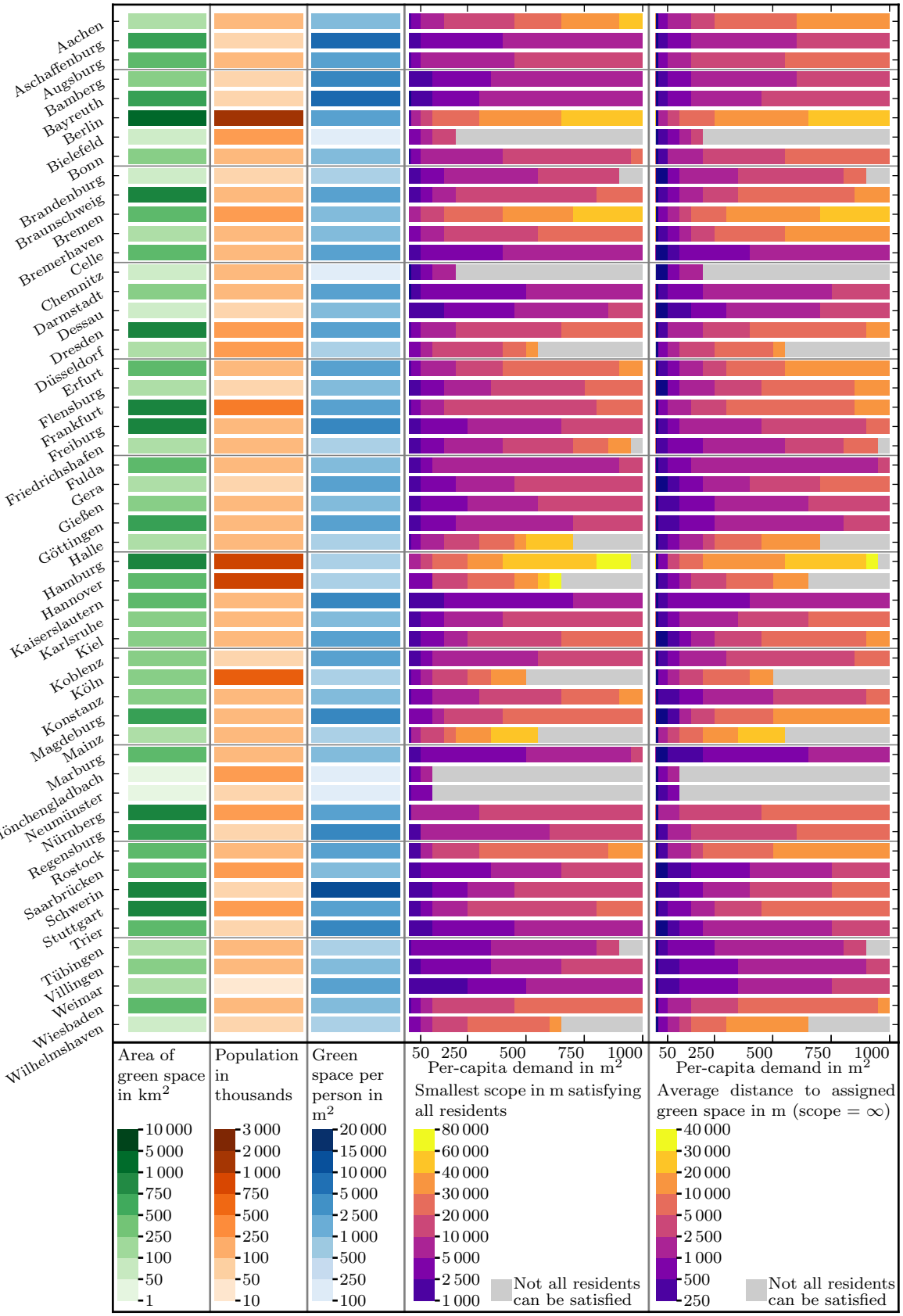


Figure 5.8: Results for 53 urban areas in Germany. The first three columns give some basic information about the urban areas while the two last columns summarize our results.

5.4.3 Evaluation

In this section, we apply our algorithm to an example and use the analysis techniques sketched in Section 5.3.3 to assess the green-space supply of urban areas.

Green-space supply of a single urban area In the following, we discuss the analysis of a single urban area. To that end, we exemplarily examine the urban area of the city of Bonn, see Figure 5.9. As a medium-sized city in Germany its extent can be printed using a resolution that allows the reader to detect and compare details of the result. Using a tool with the possibility of zooming into the map the analysis may also be done on larger cities. An interactive illustration for every scope and every considered city is available online.[5]

Figure 5.9 shows the urban area of Bonn with respect to the scopes 1 500 m, 8 000 m, and 20 000 m. For each scope we have drawn all residential areas as well as all the green spaces to which residents are assigned; all other green spaces are omitted. Consequently, with increasing scope, more green spaces are shown.

Furthermore, we color each green space with respect to its smallest relevant per-capita demand. The higher the saturation of the color of a green space, the lower is the smallest relevant per-capita demand. Hence, the saturation of the color shows the *importance* of a specific green space. Similarly, we paint each residential area with respect to its largest satisfied per-capita demand. The lighter the gray of the residential area, the lower is the highest per-capita demand for which all residents can be satisfied. Hence, light grays indicate residential areas with poor access to green spaces while dark grays indicate residential areas with easy access to green spaces.

In our model, we observe that for the scope of 1 500 m there are two regions in Bonn that have full access to green spaces only for small per-capita demands; see light gray regions in Figure 5.9. With increasing scope the green-space supply is apparently improved because the residents begin to reach green spaces further away from the city. However, for the comparatively large scope of 8 000 m, there are still residential areas that are only completely satisfied for small per-capita demands. We particularly note that our methodology is robust against small green spaces in the city center. They only impact some nearby residential areas, but do not influence the overall impression that the city center lacks green-space supply. Further, the maps indicate that the green spaces on the city's south side are particularly important: they are relevant even for small per-capita demands.

Comparing the green-space supply of multiple urban areas In our evaluation, we consider 53 cities of different size. Column 4 in Figure 5.8 shows the smallest scope that is sufficient to satisfy all residents of the considered urban area. The result of a specific urban area can be interpreted as the *robustness* of its green-space supply, which we motivate as follows. For 39 urban areas even a per-capita demand of 1 000 m^2 can be realized without leaving a resident unsatisfied. Hence, the cities' green-space supply is not exhausted even for high per-capita demands. In contrast, there are 14 urban areas whose green-space supply already collapses for smaller per-capita demands.

Considering the 39 urban areas in more detail, further differences of large extent are observable. There are 8 urban areas (e.g., *Aschaffenburg*, *Bamberg*, and *Bayreuth*) for which

[5] `http://www.geoinfo.uni-bonn.de/urbanarea`.



Figure 5.9: Analysis of the supply and accessibility of urban green space in Bonn, Germany, for various fixed scopes. An interactive illustration for every scope and every considered city is found on `http://www.geoinfo.uni-bonn.de/urbanarea`.

the scope does not need to exceed 10 km even if each resident requires 1 000 m^2. In contrast, for 16 of the 39 urban areas a scope of at least 20 km is necessary to satisfy all residents considering a per-capita of 1 000 m^2; with 48 km Berlin requires the largest scope among these cities.

Considering the 14 urban areas for which the green-space supply collapses for per-capita demands smaller than 1 000 m^2, we observe that there are urban areas whose green-space supply already collapses for rather small per-capita demands up to 250 m^2. For example, for *Neumünster* and *Mönchengladbach* a per-capita demand of 150 m^2 is not realizable without leaving residents unsatisfied. In these cases, the small scopes indicate that the diameter of the considered surrounding area is not sufficient. In contrast, there are urban areas for which the green-space supply collapses only for higher values. For *Hamburg*, for example, all residents can be satisfied up to a per-capita demand of 950 m^2. However, this requires a scope of 74 km. Hence, the robustness of its green-space supply comes at a price as a very large scope is necessary.

Column 5 in Figure 5.8 depicts the average distance to assigned green spaces with respect to the per-capita demands; in case that not all residents can be satisfied the average distance is not presented. The result of a specific urban area can be interpreted as the *accessibility* of its green-space supply, which we motivate as follows. With increasing per-capita demand, the average distance increases depending on the green-space supply of the urban area. For cities with larger easily accessible green spaces, the average distance increases more slowly than the average distance for cities with smaller ones. Hence, for the latter, the local green-space supply becomes easily insufficient for satisfying all residents. For the urban area of *Marburg*, for example, the average distance to assigned green spaces increases slower than the average distance for the urban area of *Wiesbaden*. We emphasize that both regions have a similar population size and a similar total area of green space. Still, on average, the residents of *Marburg* need to cover smaller distances than the residents of *Wiesbaden*, which implies that the green spaces of *Marburg* are more easily accessible than the green spaces of *Wiesbaden*.

Additional cluster analysis Both types of analyses can be enhanced with the cluster analysis presented in Section 5.3.3. For the analysis of a single urban area, we analyze the results for an exemplary setting with a per-capita demand of $\gamma = 50\,m^2$ which is an appropriate value according to the literature [FG09, Nat19] and a scope of 2.5 km, slightly higher than the suggestion of 1 to 2 km found in the literature [BTA+07, TES+11] such that residents with such a distance to their closest green space still contribute to the total score. Figure 5.10 depicts such a result for the city of Bonn in our model data. Here, residential areas and green spaces form 65 independent clusters. Figure 5.11 gives an overview of relative population sizes of formed clusters for all cities and, in particular, the city of Bonn (highlighted blue). In this example, less than 5% of residents have no access to green space. More than two thirds of the population are supplied within the two largest clusters (brown in Fig. 5.10, about 48% of the total population, and blue in Fig. 5.10, about 20%). These clusters correspond to parts of Bonn with the highest population density, separated by the river Rhine. Furthermore, there are 13 clusters supplying more than roughly 0.5% (1000 residents) and less than 10% of the city's population each. Seven of these minor clusters are adjacent to the large green space in the southern parts of the city, a forest. In our

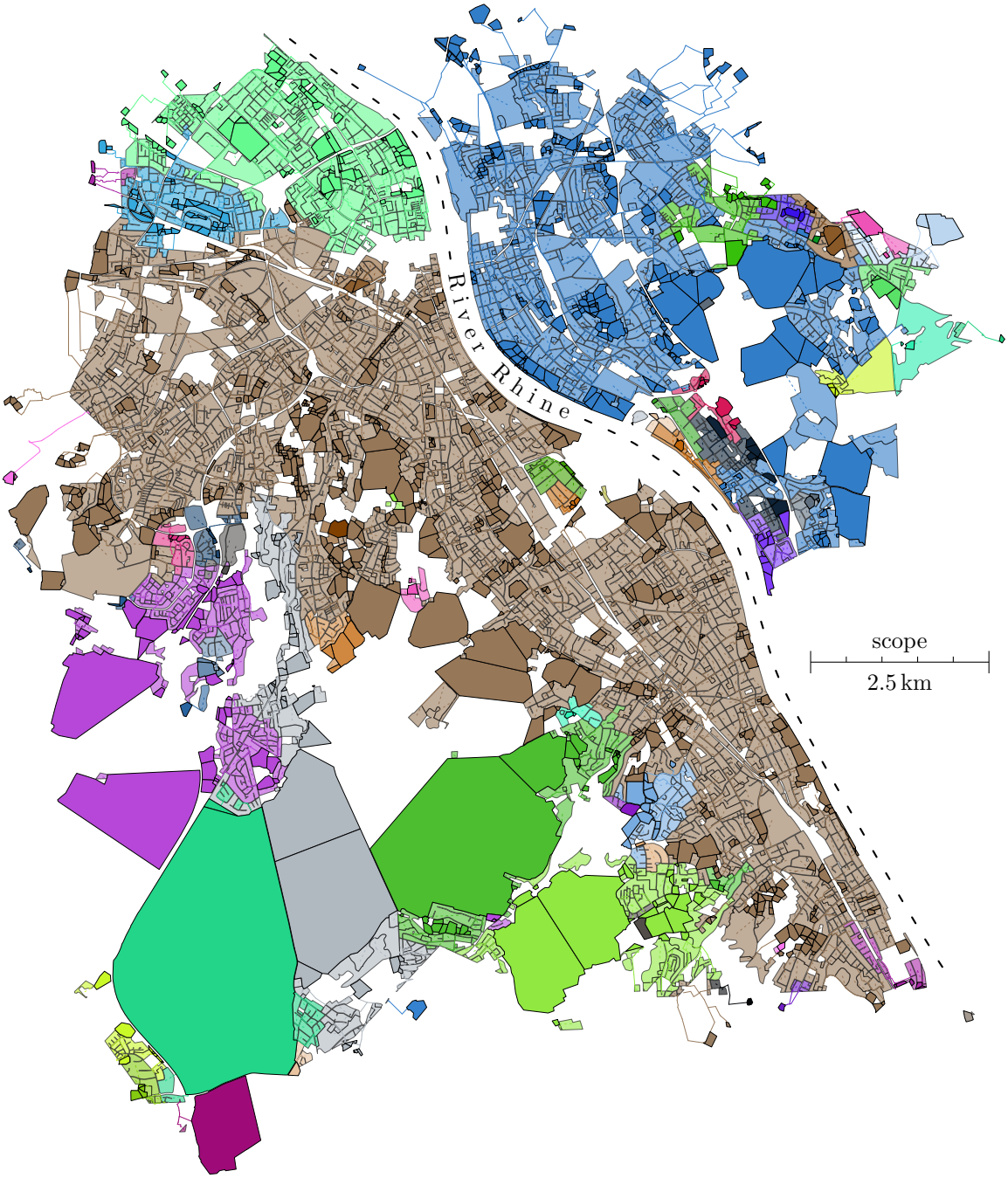


Figure 5.10: Result of the cluster analysis for the city of Bonn with $d = 2.5\,\text{km}$ and $\gamma = 50\,\text{m}^2$. Every green space, residential area and service network edge used in an optimal solution is on display. Every color represents a cluster. Polygons in bright colors represent the green spaces of the respective color; light-colored polygons represent residential areas. Segments of the service network are depicted as lines (also with bright colors).

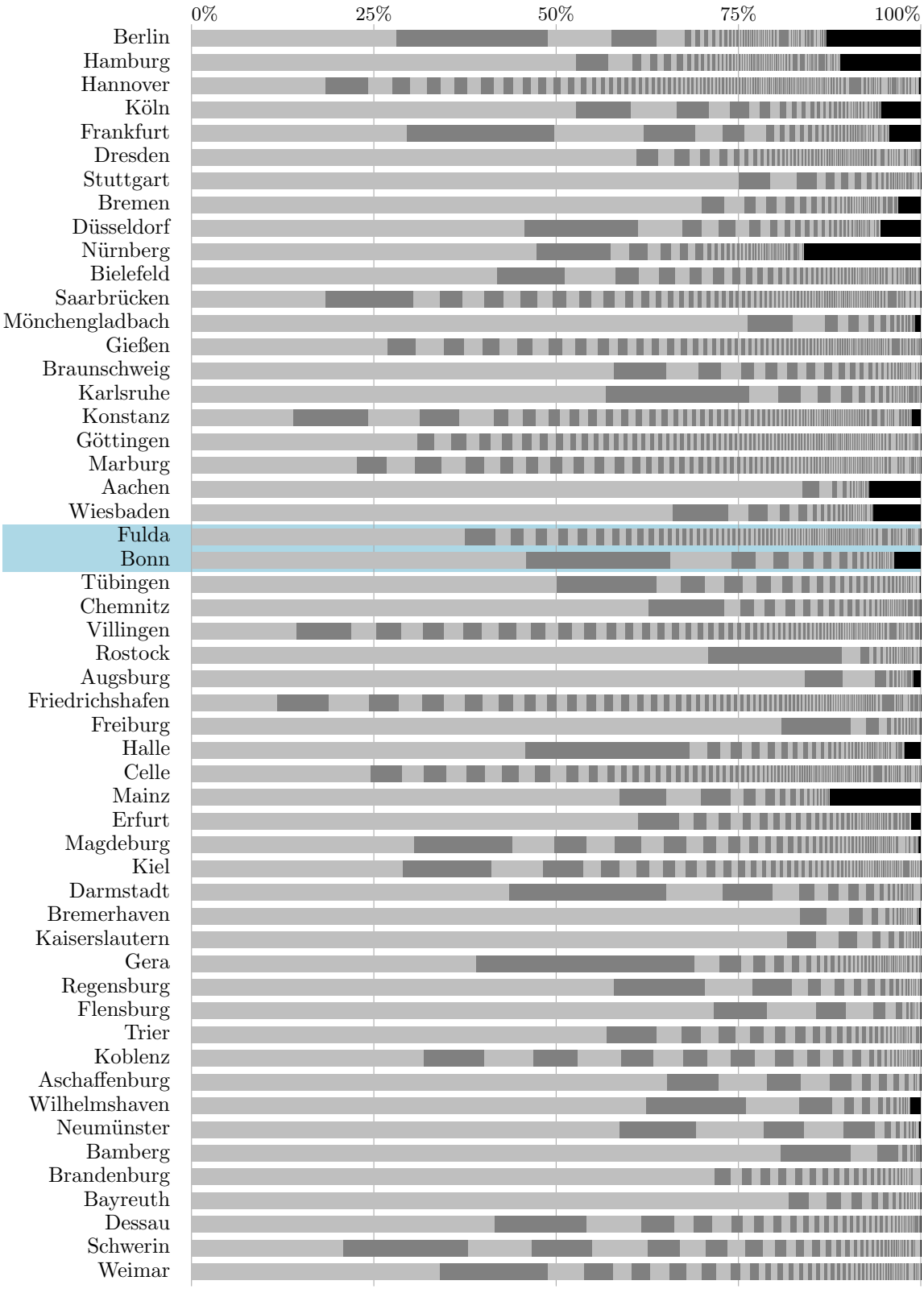


Figure 5.11: Overview of the results of the cluster analysis for all considered cities (sorted by population size in descending order) for $d = 2.5\,\mathrm{km}$ and $\gamma = 50\,\mathrm{m}^2$. Light/dark gray: Percentage of cluster population, ordered by cluster size. Black: Percentage of residents without access to green spaces in this setting.

model, these clusters satisfy more than 11% of the total population; only the vastness of the green space, with roads causing a multipart digitization, gives rise to multiple clusters. The remaining 50 clusters range from 1 to 950 residents. 53 out of 65 clusters supply less than 1% of the population each, 6.7% in total. 34 clusters supply less than 0.1% each, 1.0% in total. Most of these clusters are of no interest for a further analysis. It is up to the user to set the minimum cluster size for a closer look at the result.

The results presented in Figure 5.11 are more comprehensible when comparing entries for different cities. In our model, the data sets of Bonn and Fulda (highlighted blue in Fig. 5.11) have a similar population size. The distribution, however, differs: While the region of considered residential areas in Bonn almost coincides with the administrative boundaries of the city, for Fulda, also suburbs in the regional district are part of the data. Considering this more rural region, for Fulda, a far more local access to green spaces is detectable in Figure 5.11. Besides the larger cluster supplying roughly 35% of Fulda's population (see large blue cluster in the center of Fig. 5.12), only clusters supplying less than 5% of the population exist. The fact that there are 508 of these small clusters stresses the more local supply.

Running time A typical interactive scenario using our methodology could be as follows. The first phase is applied only once in order to create the service network at the very beginning of the scenario. Once the service network is created, its structure is not changed anymore, but the user gains the possibility of assigning attributes to each residential area and green space such as number of residents, preferences, mobility, etc. Instead of doing this only once, the user may repeatedly change the attributes to interactively explore the influence of single residential areas and green spaces. Each time, the second phase is executed. Hence, the performance of the repetitively executed second phase is clearly more crucial than the performance of the first phase. With this in mind, we have therefore focused on the second phase.

For the first phase, we put together standard algorithms without engineering their performance. For the urban area of *Berlin*, with 130 000 polygons representing green spaces, 18 000 polygons representing residential areas, and 6 million road segments our largest instance, the first phase takes about 3 minutes.

Solving the LP formulations used by far the greatest portion of the running time of the second phase. In our experiments, we measured the running time for solving $|\Gamma| \cdot |D| = 484$ LP formulations per region. Solving a single LP formulation, which we call a *run*, takes 46 seconds as a maximum and 5 seconds on average. Over 95% of all runs took at most 14 seconds. About 89% of the runs took at most 10 seconds. These running times indicate that our approach does not allow real-time animations, but is suitable for interactive systems where the user can update the assignment on demand. Apart from interactive systems, our approach can also be used for the systematic and automatic evaluation of green spaces with different parameter settings. Accumulating the running times of all runs of a single urban area in our experiments yields 3.3 hours in maximum and 40 minutes on average. In total, 35 hours were necessary to process all 53 cities.

For the third phase, concerning the running-time analysis, we limit ourselves to the processes extracting information since the time cost for visualization depends on the kind of

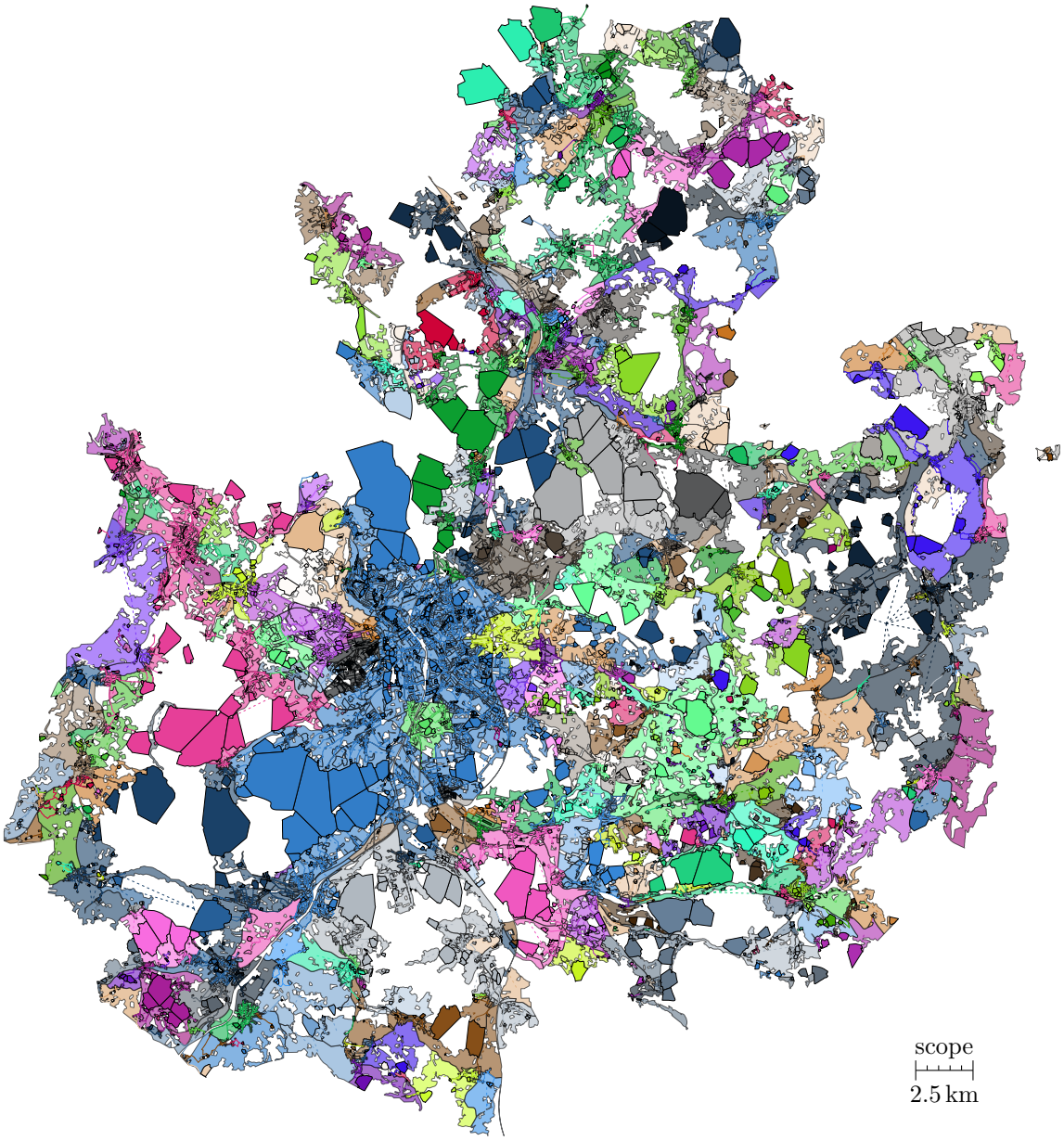


Figure 5.12: Result of the cluster analysis for the city and the rural district of Fulda with $d = 2.5\,\text{km}$ and $\gamma = 50\,\text{m}^2$. Every green space, residential area and service network edge used in an optimal solution is on display. Every color represents a cluster. Polygons in bright colors represent the green spaces of the respective color; light-colored polygons represent residential areas. Segments of the service network are depicted as lines (also with bright colors).

presentation of the data. The three of the four measures we propose depend on either the residential areas or the green spaces only. Thus, for a single city, their computation can be done in $O(|R| \cdot |\Gamma| \cdot |D|)$ or $O(|G| \cdot |\Gamma| \cdot |D|)$ time, respectively. Merely for the computation of the average distance to assigned green spaces, it is necessary to take a look at the service network. Like for computing the connected components, a running time in $O(|E \cup F| \cdot |\Gamma| \cdot |D|)$ results. In our experiments, this post-processing step took us far less than 1 second for a single run. The third phase for the 484 runs of *Berlin* combined took us 2 minutes in total, i.e., less than 6 seconds per run (roughly 1% of its running time).

5.5 Conclusion

We presented a highly general model for the evaluation of green spaces in urban areas. Our approach is based on the idea of finding an assignment of the residents to green spaces that maximizes a predefined score while capacity constraints for the green spaces are respected. In a specialization we described, this score depends on the distances the residents have to overcome in a solution as well as predefined parameters such as the attractiveness of green spaces. This way, the accessibility of urban green spaces is examined. Limiting the flow of residents to a green space with its capacity allows us to simultaneously take the green-space provision in an urban area into account. This specialization of the model allowed us to transfer the model to a road network which improved its performance due to a reduced complexity of the model (linear in the size of the network versus quadratic in the number of green spaces and residential areas as it occurs in the basic model). This advantage makes the analysis of metropolitan areas like Berlin possible. Besides a small running example, we also contributed a detailed description of the workflow and the application of the model. As demonstrated, our methodology can be used for analyzing particular urban areas as well as a set of urban areas in general. Apart from yielding abstract parameters describing the green-space supply, our approach allows the user to run spatial analyses on the level of single residential areas and green spaces. A discussion panel with domain experts from urban planing approved that our approach will be of great use for urban planning for both easily assessing existing green spaces and planning future land usage. Especially, the methodology is useful in interactive scenarios for urban planning. By means of our approach, an urban planner may interactively explore the influence of potential residential areas, green spaces, and roads using maps such as in Figure 5.9. They may change the importance of green spaces or even introduce new regions. Each time, our model is updated and the result is visualized. Thus, the user can easily assess the impact of the changes made.

Furthermore, with the clustering, we present a tool for detecting patterns in the distribution of green spaces. Results of analyses on city scale yield additional information which parts of cities share green spaces. The comparative analyses give also information about the distribution of the green-space supply: Diagrams such as the one in Figure 5.11 make it easy to see which city has a large pool of green spaces for the general public and which cities provide their residents with green spaces on a more local basis.

The tools we introduce are presented with a very generic setup. This generality, however, provides users with many possibilities to adapt the tools to their use case. Among others, the following specializations and research questions arise.

- We kept our experiments simple to evaluate the core of our methodology. In practice, it lends itself to use a more complex parameterization reflecting reality more accurately. If users have more information about the road network at their disposal, it is desirable to change the edge weight in the road network from geodesic distance to travel times. Furthermore, with additional information about the population structure of residential areas or the attractiveness of green spaces, the model can be enhanced by adapting the weights α_r and β_g, respectively. Above all, the results depend on a appropriate choice of input data. In our model data, the recreational value of lakes, river banks, open spaces, and so on did not get their fair share.
- An interesting followup question is to analyze the utilization of the road network in detail. Which roads are used more than others? May these insights help in traffic planning, especially for weekends? A closer look at the computed flow may give insights.
- The network-based model anonymizes the assignment in the sense that we cannot keep track of single residents. As presented in this work, the network-based model in general allows the user only to assign residents of specific residential areas to a set of green spaces. In order to obtain information about the exact assignment of residents of specific residential areas to green spaces falling back to the more time-consuming basic model, presented early in Section 5.3.1, is necessary. Nevertheless, the network-based model with the clustering may be applied in a preprocessing step: To reduce computation time, it is helpful to know in advance which combination of green spaces and residential areas is plausible.
- Conversely, for certain applications, our model might provide too detailed information with respect to privacy. Although we have no problematical application on our minds, we want to refer to research on anonymization at this point [ILGZ14].
- Based on our approach, an evaluation of the accessibility of public services may be possible. In order to examine the coverage of, for example, hospitals, medical practices, schools, playgrounds, etc., further research is necessary to determine which requirements the suppliers, residential areas, or the road network have to meet.

In particular with regard to the possibilities of adapting our approach, there is, on the downside, the limitation that our score function which expresses the total benefit of green spaces decreases linearly with an increasing distance between residential areas and green spaces which may not be appropriate. However, we present a conceptually simple tool set for evaluating the green-space supply of a city considering both green-space provision and accessibility in a combined manner. Besides, our approach yields a globally optimal solution that can be found efficiently.

6 Inferring routing preferences of bicyclists from sparse sets of trajectories

The following chapter is mainly taken from a joint work with Alina Förster, David Schunck, Youness Dehbi, Ribana Roscher and Jan-Henrik Haunert [OFS+18]. The presented work won the best-paper award at the International Conference on Smart Data and Smart Cities (SDSC) in 2018.
In this chapter, we present an algorithm for adjusting routing models to specific groups of bicyclists based on sparse sets of trajectories. First, applying an established algorithm, we cluster these trajectories to obtain user groups (in our example: `mountainbiking`, `racingbiking`, and `biking`). Based on this clustering, we determine group profiles concerning road-type preferences, which form the foundation for the following step: We apply an algorithm for aggregating routing criteria in order to merge road-type preferences and the demand for short paths to a single criterion. This criterion is the result of our algorithm, our recommendation for a routing criterion for the respective group of bicyclists. Furthermore, we use the results of this criteria-aggregating algorithm to validate the previously performed grouping of bicyclist.
So far, the algorithm applied in the final step has been published only as an outline in an early stage [ONH17]. Consequently, a detailed description of this algorithm can be found in the appendix of this chapter, see Section 6.A.

Abstract
Understanding the criteria that bicyclists apply when they choose their routes is crucial for planning new bicycle paths or recommending routes to bicyclists. This is becoming more and more important as city councils are becoming increasingly aware of limitations of the transport infrastructure and problems related to automobile traffic. Since different groups of cyclists have different preferences, however, searching for a single set of criteria is prone to failure. Therefore, in this paper, we present a new approach to classify trajectories recorded and shared by bicyclists into different groups and, for each group, to identify favored and unfavored road types. Based on these results we show how to assign weights to the edges of a graph representing the road network such that minimum-weight paths in the graph, which can be computed with standard shortest-path algorithms, correspond to adequate routes. Our method combines known algorithms for machine learning and the analysis of trajectories in an innovative way and, thereby, constitutes a new comprehensive solution for the problem of deriving routing preferences from initially unclassified trajectories. An important property of our method is that it yields reasonable results even if the given set of trajectories is sparse in the sense that it does not cover all segments of the cycle network.

6.1 Introduction

Faced with the problem of organizing the traffic in rapidly growing cities, many city planners try to support cycling as an environmentally friendly and healthy means of transport. In order to increase the attractiveness of cycling, methods for analyzing the routes that bicyclists prefer are needed. Spatial information that is collected by volunteers (i.e., volunteered geographic information) can be used to establish a rich data basis for such methods. In particular, trajectories that cyclists record and share via on-line platforms (e.g., GPS tracks from Strava or GPSies) provide information that is not available from other sources. Extracting the information about routing preferences in a meaningful form is far from trivial, however, since the data sets have to be subdivided (e.g., to analyze routing prefcerences separately for different groups of cyclists) or integrated (e.g., to enrich GPS tracks with information on road types). Therefore, a methodology that combines multiple data sources and algorithms is needed. A problem that has not been sufficiently addressed yet is how routing preferences can be inferred if the given set of trajectories is *sparse* in the sense that the trajectories do not cover all segments of the cycle network – see the extract from the input data that we used in our experiments in Fig. 6.1. Still, it is a desirable goal to learn the routing preferences of an individual or a group of cyclists in a form that allows the computation of an optimal path between any two locations in the network. In this paper, we present a new methodology to achieve this goal.



Figure 6.1: The road segments for a small part of our test area, classified into segments that were used by at least one trajectory (black) and those that were not used (gray). The latter includes 75.81% of all edges and 68.70% of their total length.

Our methodology for inferring routing preferences of cyclists from multiple sources requires trajectories (e.g., GPS tracks), land-use information, and a road network model in the form

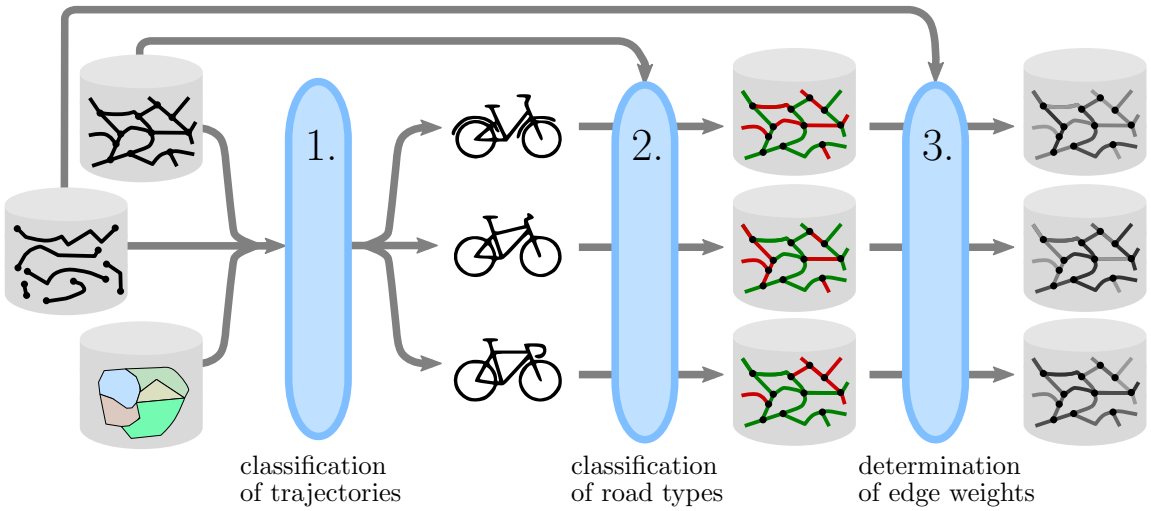


Figure 6.2: An overview of our method for the identification of cyclist groups and their routing preferences. After a data acquisition from different sources, a multi-source data analysis consisting of three steps is performed.

of a graph $G = (V, E)$ as input – the latter should include relevant bicycle paths as well as information on road types, such that roads that are forbidden for cyclists (e.g., motorways) can be removed and influences of road types on route choices can be analyzed. The ultimate goal is to determine an edge weighting $w\colon E \to \mathbb{R}_{\geq 0}$ for each individual cyclist, to reflect his or her personal routing preferences, or at least one edge weighting for each group of cyclists (e.g., mountain bikers, racing cyclists, and others) that we can identify with the available data. Generally, in the context of this paper, the weight $w(e)$ of an edge $e \in E$ is interpreted as the cost for traversing e – the edge weights are equal to the lengths of the edges if the cyclists simply prefer short routes, but other weight settings are needed to express that cyclists accept detours in order to avoid unfavorable road segments (e.g., unpaved trails in the case of racing cyclists). We also write $w(P)$ to refer to the total weight of a path P. Since a weighted graph model is required as input by most routing algorithms, the outcome of our method can be used to infer user-dependent or group-dependent optimal paths between any two locations in the cycle network. This could be useful for cyclists who use bicycle navigation systems for route planning, but also for spatial planners who conduct shortest-path analyses with geo-information systems, e.g., to predict traffic loads for planned bicycle paths.

Three steps are conducted in order to get from the source data to the weighted graph models. These steps are illustrated in Fig. 6.2 and specified below.

1. In the first step, the different information sources are combined using a map-matching algorithm. Geometric buffering operations as well as shortest-path computations (with a default weight setting) are applied to extract meaningful features for an unsupervised classification method. This is used to classify the trajectories with respect to different cyclist groups.

2. In the second step, favored and unfavored road types are identified for each group. In this context, a high proportion of a certain road type within the trajectories of a group is an indicator of preference of this road type.
3. In the third step, we learn a function that maps edge types and edge lengths to edge weights. Note that this function yields a weight for *every* edge, no matter whether or not it is used by any of the trajectories. The edge weights can be used to consider the routing preferences when computing new routes. We will show how to conduct this step for each single trajectory as well as for the set of trajectories of each cyclist group.

The main contribution of this paper is the automatic identification of a method for identifying favorable and unfavorable road types and the computation of a weighted graph model for a group of users that reflects the preferences of the road types.

The remainder of this paper is structured as follows. We review related work in Sect. 6.2 and present our methodology in detail in Sect. 6.3. Then, we discuss our experimental results in Sect. 6.4 and conclude the paper in Sect. 6.5.

6.2 Related work

Analyzing trajectories has become a major research discipline within computer science and geographic information science – we do not aim to give a comprehensive overview but refer to the survey article by Mazimpaka and Timpf [MT16]. An important task of trajectory analysis is to improve navigation systems and routing algorithms based on trajectories recorded by users. Based on taxi trajectories, Yuan et al. [YZZ+10] were able to infer travel times for road segments and, thus, to enable the computation of fastest routes for cars. More generally, trajectories can be used to augment a network model with additional attributes acquired by users, for example, with information on road surface quality inferred from accelerometer data of bicyclists [RSD+10]. Kessler [Kes13] and Sultan et al. [SBHD17] have discussed in detail how volunteered geographic information (VGI) can be used to analyze bicycle routes. An open problem is still, however, to learn previously unknown routing criteria and to adopt them for the application in routing algorithms.

In the context of understanding cyclists' behavior and their routing preferences, Broach et al. [BDG12] observed 164 cyclists over a couple of days. They analyzed their practices based on recorded GPS trajectories. The distance, turn frequency, slope, intersection control and traffic volumes turn out to be the major parameters influencing the choice of the cyclist trip paths. Infrastructures such as off-street bike paths as well as the trip category, e.g. commute or utilitarian, have also an impact on the route preferences of the investigated tracks. In contrast to Broach et al. [BDG12], we are particularly interested in analyzing how route choices are influenced by road types. Furthermore, we aim at the automatic identification of cyclist groups from crowd-sourced data and the learning of a routing model for each detected group based on its specific preferences.

In order to derive a weighted graph model reflecting the routing preferences of cyclists, Bergmann and Oksanen [BO16] have chosen a rather pragmatic approach by counting for

each road segment the number of users and the number of trajectories using it. Based on these numbers, they have defined three different measures to derive edge weights, all of which are based on the assumption that highly used road segments should receive low weights. We argue, however, that the frequency of usage should not directly be translated into an edge weight. Consider for example a triangular graph whose edges represent connections between three cities A, B, and C. If we observe a large amount of traffic on the edge $\{A, B\}$ connecting A and B, we must not conclude that this edge is by any means 'cheaper' than $\{B, C\}$ or $\{C, A\}$ and that it should receive a low weight. Instead, the high usage of edge $\{A, B\}$ might also be due to the fact that many people commute between A and B. Furthermore, inferring the weights of edges from the frequency of their usage fails if the set of trajectories is sparse.

Probably the first method that infers routing preferences from a sparse set of trajectories has been presented by Balteanu et al. [BJS13]. As all methods that we review in the following, it even works if only a single trajectory is provided as input. More precisely, given a graph G with two edge weightings w_0, w_1 (e.g., travel time and geometric distance) and a user's path P (the trajectory) between two vertices s and t in G, the aim is to infer a parameter β such that an s-t-path P' minimizing

$$\max\{\beta \cdot w_0(P'), (1-\beta) \cdot w_1(P')\} \tag{6.1}$$

is most similar to P. Without reviewing in detail how similarity is defined in this context, we note that the parameter β inferred by the method can indeed explain how the user trades off between w_0 and w_1. A clear disadvantage of the method of Balteanu et al. [BJS13] with respect to its practical relevance is, however, that the trained routing model is of little use if the aim is to compute new routes with standard routing algorithms (e.g., the algorithm of Dijkstra [Dij59]) which, usually, require a single edge weighting as input. More precisely, a standard routing algorithm does not yield a path P' minimizing (6.1) for a trained parameter β. Therefore, in the following, we focus on methods that try to define a new edge weighting w based on a linear combination of the given edge weightings. After the coefficients of the linear combination have been learned and, thus, the new edge weighting is fixed, one can use standard routing algorithms to compute routes that are optimal with respect to w.

Funke et al. [FLS16] have studied the problem in which a graph G with multiple edge weightings $w_1, \ldots, w_d$ as well as a path P between two vertices s and t in G are given as input and the aim is to compute a linear combination $w = \alpha_1 \cdot w_1 + \ldots + \alpha_d \cdot w_d$ of the weightings such that P is a weight-minimal s-t-path with respect to the new weighting w. This weighting is assumed to represent the routing preferences of a user who chose P as his or her route. Unfortunately, the problem can be infeasible for a path corresponding to the trajectory of a user, since the path may not be optimal with respect to any weighting. Funke et al. address this issue by suggesting that if the problem is infeasible for a given path then the path should be divided into two subpaths of equal length and the problem should be solved independently for each of the two subpaths (which many require further recursive splitting to end up with feasible problem instances). With this approach, however, artificial split points are introduced and different linear combinations are obtained for the different subpaths.

The algorithm of Oehrlein et al. [ONH17] is similar to the one of Funke et al. [FLS16] in the sense that it computes a partition of a given path into multiple subpaths and a linear combination of different weightings. However, the partition and the new weighting are computed such that *all* of the resulting subpaths are optimal with respect to the *same* weighting w, meaning that the different subpaths are not considered as independent problem instances. Furthermore, instead of partitioning a path into two subpaths of equal lengths, the algorithm of Oehrlein et al. uses an optimization criterion to decide where to introduce split points. More precisely, given a graph G with two edge weightings w_0 and w_1 and a path P in G, the algorithm yields a new weighting $w_\alpha = (1-\alpha) \cdot w_0 + \alpha \cdot w_1$ and a partition of P into a *minimum number* of subpaths such that each of the subpaths is optimal according to the weighting w_α. Compared to the algorithm of Funke et al., the algorithm of Oehrlein et al. is certainly more advanced with respect to how it computes the split points[1], but it has the disadvantage that it can deal with only two given weightings w_0 and w_1 and not with an arbitrary number d of weightings. Nevertheless, we choose this method since inferring a trade off between two criteria from a sparse set of trajectories is already challenging. In the following, we refer to the split points computed by the method as *milestones* and the partition of the given path induced by the split points as a *milestone segmentation*.

An important property of the method of Oehrlein et al. is that it not only computes the parameter α corresponding to an optimal milestone segmentation but that it systematically explores different values for α and tests the effect on the size of the milestone segmentation. This offers new possibilities of studying the quality of a bi-criteria routing model as a function of its trade-off parameter α, which we will show in Sect. 6.4 for the experiments that we conducted.

6.3 Methodology

In this section we present the mathematical foundations of our method, including the routing model whose parameter we aim to learn (Sect. 6.3.1) and the concepts behind each of the three steps of our method (Sections 6.3.2–6.3.4).

6.3.1 Routing Model

A meaningful representation of a user's routing preferences in a given graph $G = (V, E)$ is a weighting $w\colon E \to \mathbb{R}_{\geq 0}$ that assigns to each edge $e \in E$ a weight $w(e)$. This can be assumed to represent a cost for traversing edge e. Our aim is to learn such a weighting from trajectories, which will allow us to compute optimal paths for a user or group of users with known algorithms, for example, with the classical algorithm of Dijkstra [Dij59] or with the help of modern speed-up techniques, such as contraction hierarchies [GSSD08].

Since the given graph G may not be completely covered by the trajectories, the trajectories alone do not suffice to infer weights for all edges of G. Therefore, a good strategy is to define

[1] In fact, the approach of Funke et al. eventually provided inspiration to improve our algorithm. In Section 6.A, we describe an advanced version of our algorithm based on the work of Funke and Storandt [FS13] and Funke et al. [FLS16].

the weighting based on attributes that are given for each edge (for example, its length and road type) and to use the trajectories only to infer a small number of parameters that condense the attributes into a weight. In this paper, we suggest a model that requires for each user group a classification of road types into unfavorable and favorable types (for example, arterial streets and bicycle paths, respectively) and, additionally, a single parameter $\alpha \in [0, 1]$. According to this model, the weight of edge $e \in E$ is

$$w(e) = \begin{cases} \alpha \cdot \text{length}(e) & \text{if} \quad e \in E^+ \\ (1-\alpha) \cdot \text{length}(e) & \text{if} \quad e \in E^- \end{cases} \tag{6.2}$$

where length(e) is the length of e and $\{E^-, E^+\}$ is a binary classification of E into a set E^- of edges with an unfavorable road type and a set E^+ of edges with a favorable road type. This model implies that traversing an edge with an unfavorable type is by factor $\frac{1-\alpha}{\alpha}$ more expensive than traversing an edge of the same length with a favorable type. Obviously, one would expect $\alpha \leq 0.5$, since otherwise edges with unfavorable types would be preferred, which would be a contradiction. However, we leave it to the inference algorithm that we apply (see Sect. 6.3.4) to select $\alpha \in [0, 1]$ and, afterward, test for $\alpha \leq 0.5$ to check the consistency of the result. To summarize, we need to detect different groups of users and for each group the binary classification $\{E^-, E^+\}$ as well as the parameter α.

Obviously, this approach could be generalized by classifying the road types into more than two classes. The more classes are considered, however, the more parameters would have to be learned in order to derive the edge weights from the types and the geometric lengths of the edges. Since many road segments are not covered by any trajectory and since for some road types only few road segments exist, inferring a binary classification and learning the parameter α for each user group is already challenging. Nevertheless, learning a more sophisticated model is an interesting task for future research. Since the algorithm of Oehrlein et al. [ONH17] that we apply in our workflow is currently limited to two edge weightings, however, such an improvement would require a more substantial innovation.

6.3.2 Classification of Trajectories

In the first step of our method, a multi-source data analysis is performed in order to automatically classify the set of trajectories with respect to different cyclist groups. To this aim, openly accessible GPS-tracks are collected from a user-driven platform. The trajectories are then augmented by additional information such as road types. The extraction of additional features is performed after a map-matching process, which establishes correspondences between a given trajectory and an underlying road network of the region of interest. In order to achieve an accurate analysis, the trajectories are also enriched by information about the surrounding areas stemming from a digital landscape model. This, in particular, gives insight into the land-use categories of the areas through which the trajectories pass. Furthermore, for each trajectory, a path of minimum length is computed connecting the trajectory's source and destination. This yields additional interesting features, such as the length ratio between the trajectory and the optimal path (also known as the detour factor or dilation). All this information is serving as a rich feature set for the extraction of cyclist groups in an unsupervised learning process.

The classification of the trajectories into meaningful cyclist groups is done in an unsupervised way using the k-means algorithm [Llo82]. Although the users normally specify the types of their trajectories when they share them and, thus, a user-specified grouping of the trajectories is available, the assignment is subjective and partly erroneous, which is underlined by our experiments (see Sec. 6.4.2). Therefore, in the subsequent steps of our method, we use the result of the unsupervised classification algorithm instead of the user-specified types.

6.3.3 Recognizing Unfavorable and Favorable Road Types

Road networks are usually represented as sets of line segments with associated road types. Therefore, as a result of the map-matching process, we obtain for each user group the distribution of road types over the total length of the trajectories. Although such statistics give interesting insights, we have to be careful of what we conclude. Suppose that the type "residential street" constitutes 95% of the total length of a group's trajectories. This high percentage may either be due to the fact that the group considers residential streets as favorable or because there is a lack of bicycle-friendly paths and, thus, the users had to choose an unfavorable road type. Therefore, we compute for each trajectory the geometrically shortest path in the road network connecting the trajectory's start and end point and use that path as a reference. For each road type c, we compare the relative share $r_{\text{user}}(c)$ of c among the total length of the trajectories with the relative share $r_{\text{shortest}}(c)$ of c among the total length of the shortest paths. If $r_{\text{user}}(c) > r_{\text{shortest}}(c)$, one may argue that the users had the possibility of using shorter paths but decided to use longer paths with a larger share of type c. This can be seen as an indication of c being a favorable type. Consequently, we define

$$E^+ = \left\{ e \in E \middle| \, r_{\text{user}}(c(e)) \geq r_{\text{shortest}}(c(e)) \right\}, \tag{6.3}$$

$$E^- = \left\{ e \in E \middle| \, r_{\text{user}}(c(e)) < r_{\text{shortest}}(c(e)) \right\}, \tag{6.4}$$

where $c(e)$ is the road type of edge e.

6.3.4 Inferring Edge Weights

Generally, an edge weighting w alone can not explain the trajectories of a user group since, for example, even within one group different criteria are applied or the trajectories include round trips that were clearly not chosen as minimum-cost paths between two vertices. Moreover, the model that we introduced with Equation (6.2) may be too restrictive to subsume the weighting actually applied by a user. Nevertheless, we aim to determine the parameter α such that the model explains the trajectories of a user group *as much as possible*. For this purpose, we apply the algorithm by Oehrlein et al. [ONH17]. Recall that, given a user's trajectory T as a path in a graph $G = (V, E)$ with two edge weightings w_0 and w_1, this algorithm partitions T into a minimum number of sub-trajectories and, simultaneously, selects a parameter $\alpha \in [0, 1]$, such that each of the resulting sub-trajectories is an optimal path in G, in the sense that no path connecting the same two vertices is better

according to the weighting $w = \alpha \cdot w_0 + (1 - \alpha) \cdot w_1$. Since minimizing the number of sub-trajectories is the same as maximizing their average length, the weighting w that is learned with the method explains the routes chosen by the users relatively well.

Oehrlein et al. [ONH17] used their algorithm to understand how slope affects the route choice of bicyclists. With our model, however, where the weighting w should reflect unfavorable and favorable road types, the algorithm needs to be applied with the following setting:

$$w_0(e) = \begin{cases} \text{length}(e) & \text{if} \quad e \in E^+ \\ 0 & \text{if} \quad e \in E^- \end{cases} \tag{6.5}$$

$$w_1(e) = \begin{cases} 0 & \text{if} \quad e \in E^+ \\ \text{length}(e) & \text{if} \quad e \in E^- \end{cases} \tag{6.6}$$

With this setting, $\alpha \cdot w_0 + (1 - \alpha) \cdot w_1$ is indeed equal to w as defined in Equation (6.2).

The algorithm of Oehrlein et al. [ONH17] requires integer weights as input, which we ensure by rounding the edge lengths to m. It works by systematically testing different values $\alpha \in [0, 1]$, including the interval boundaries 0 and 1. We encountered very long running times for those extremal values and, therefore, decided to restrict the search to $\alpha \in [0.1, 0.9]$. With this we still take into consideration that, in order to avoid an unfavorable edge $e \in E^-$, a user may accept a detour of nine times the length of e. However, longer detours are not considered.

6.4 Experiments

This section presents our conducted experiments and experimental results and gives insight into the data used in the different steps of our approach.

6.4.1 Data

Since Bonn is representing an example of a bicycle-friendly city, we decided to demonstrate our approach for this region. 82% of the households in Bonn are owning at least one bicycle. Furthermore, not only the city but also the surrounding areas, for instance "Siebengebirge" and the bank of the Rhine river, are attractive for bicycle tours. The Bonn's city council is striving till 2020 to declare Bonn as the capital of bicyclists in the federal land of North Rhine Westphalia.

Our experiments are performed on crowd-sourced data stemming from the user-driven platform GPSies[2]. From this platform, GPS trajectories, which have been recorded by users with different preferred activities, can be downloaded. In our context, we are especially interested in the following three types of cyclist activities: `biking`, `mountainbiking`, and `racingbiking`. For the evaluation of our algorithm, we downloaded about 250 trajectories for each user group from the region of Bonn and surroundings in Germany. Beside the GPS-coordinates, each track contains additional information about the whole length, climb and

[2] `https://www.gpsies.com/`

descent of the trajectory. Furthermore, two types of tracks are discriminated: circular and simple tracks. We denote these features as `feature set a`.

In order to analyze these trajectories, we extracted additional features from road segments corresponding to the underlying trajectories. The correspondences were computed with the map-matching algorithm of Haunert and Budig [HB12]. To this aim, we used a road network of the same area from `OpenStreetMap`[3] (`OSM`). We denote these features as `feature set b`.

In this way, each trajectory path is augmented by the information acquired from the associated road segment from `OSM`. For our purpose, the street type category including among others roads, paths and cycle tracks is of great relevance. The inferred information enables for example a trajectory analysis depending on the used street types for each cyclist group.

In order to learn different weightings for different user groups of cyclists, additional information about a given trajectory and its surrounding is needed. Thus, we exploited data related to our region of interest stemming from the German Digital Landscape Model ATKIS-DLM[4]. The latter is an object-based vector model which defines an object set with several object types accordingly. The object types comprise for instance *woodland*, *arable land* and *settled land*.

6.4.2 Results of the Trajectory Classification

In this experiment, we classify the trajectories into specific activity groups and compare them to the user-provided groups. For this, we use the provided information from both `feature set a` as well as `feature set b`. The features are z-normalized to zero mean and unit standard deviation to ensure an equal weighting of each single feature.

We cluster the data utilizing k-means and manually assign activity groups to the resulting clusters. We run k-means with different initializations and choose the result with the highest compactness. We choose different numbers of clusters, and manually decide on the best number by means of the quality of the assignment.

Furthermore, we determined the importance of specific features using the reliefF algorithm [Kon94] and analyzed the influence of restricting the set of features to the most important ones on the k-means clustering result. We manually tested several values for the amount of neighbors necessary to calculate the importance for each feature, and report the results for $k_{\text{reliefF}} = 100$. We observed that for larger values the set of the most important features converges to a fixed set. We used the calculated weights and determined all features that lie within the 90%-quantile.

The evaluation of different numbers of clusters confirm the user-provided groups such that k-means provide the best interpretable result using three clusters with features which can be assigned to the user-provided groups. Table 6.1 shows the contingency table, which is a detailed analysis of the number of trajectories assigned to the three different groups `racingbiking`, `mountainbiking`, and `biking` by the user and by k-means. The table also includes information about the number of trajectories assigned to the same group and assigned to different groups by the user and by k-means.

[3] `https://www.openstreetmap.org/`
[4] `https://www.opengeodata.nrw.de/`

clustered \ original	mountainbiking	racingbiking	biking	sum
mountainbiking	125	20	39	184 (68%)
racingbiking	10	135	47	192 (70%)
biking	17	63	141	221 (64%)
sum	152 (82%)	218 (62%)	227 (62%)	597 (67%)

Table 6.1: Contingency table, showing the number of trajectories assigned by the user and by k-means clustering to `biking`, `mountainbiking` and `racingbiking`.

Overall, there is a consensus in 67% of all trajectories. There is an increase for `mountainbiking` and a decrease for `racingbiking` after k-means is applied. Although nearly the same numbers of trajectories are assigned to `biking`, this group shows the largest difference in our comparison. Around 40% of users which assign themselves to the group `biking` are classified as a different group by k-means. Especially users who classify themselves as part of the group `racingbiking` are assigned to `biking` by k-means. For a more detailed examination, we choose different trajectories which have a different assignment by the user and by k-means, and analyzed them by means of various features. It turned out that in most cases users assign themselves to an activity group which does not fit their biking behavior, or the trajectory's features lie close to the cluster boundary.

Finally, we analyzed the features' importance obtained by reliefF. The sorted list of the most important features in decreasing order is (1) the route type (circular or simple track), (2) the altitude range, (3) the difference between the length of the actual trajectory and the shortest path-trajectory, (4) percentage of agricultural area close the trajectory, (5) percentage of forest close the trajectory, followed by multiple features defining the road type, and the living environment. We repeated k-means clustering with the most important features and compared it to the clustering results using all features. Both clustering results agree in 96% of all trajectories. Moreover, we compared the contingency table obtained by k-means with all features (compare Table 6.1) and the contingency table obtained by k-means with the most important features, and receive a mean absolute difference of 3.35%. Both result indicate that the identified most important features describe the activity groups well.

6.4.3 Results of the Road-Type Classification

In this experiment we compute for each trajectory T a shortest path P in the road network that connects the start vertex and end vertex of T. For each group of bicyclists, we analyze the share of the different road types among the total length of all trajectories as well as among the total length of all shortest paths. A comparison allows us to infer which of the road types are favored and unfavored.

Figure 6.3 summarizes the share of each road type among the total length of the trajectories (i.e., the actual routes of the users) and among the total length of the shortest routes. The

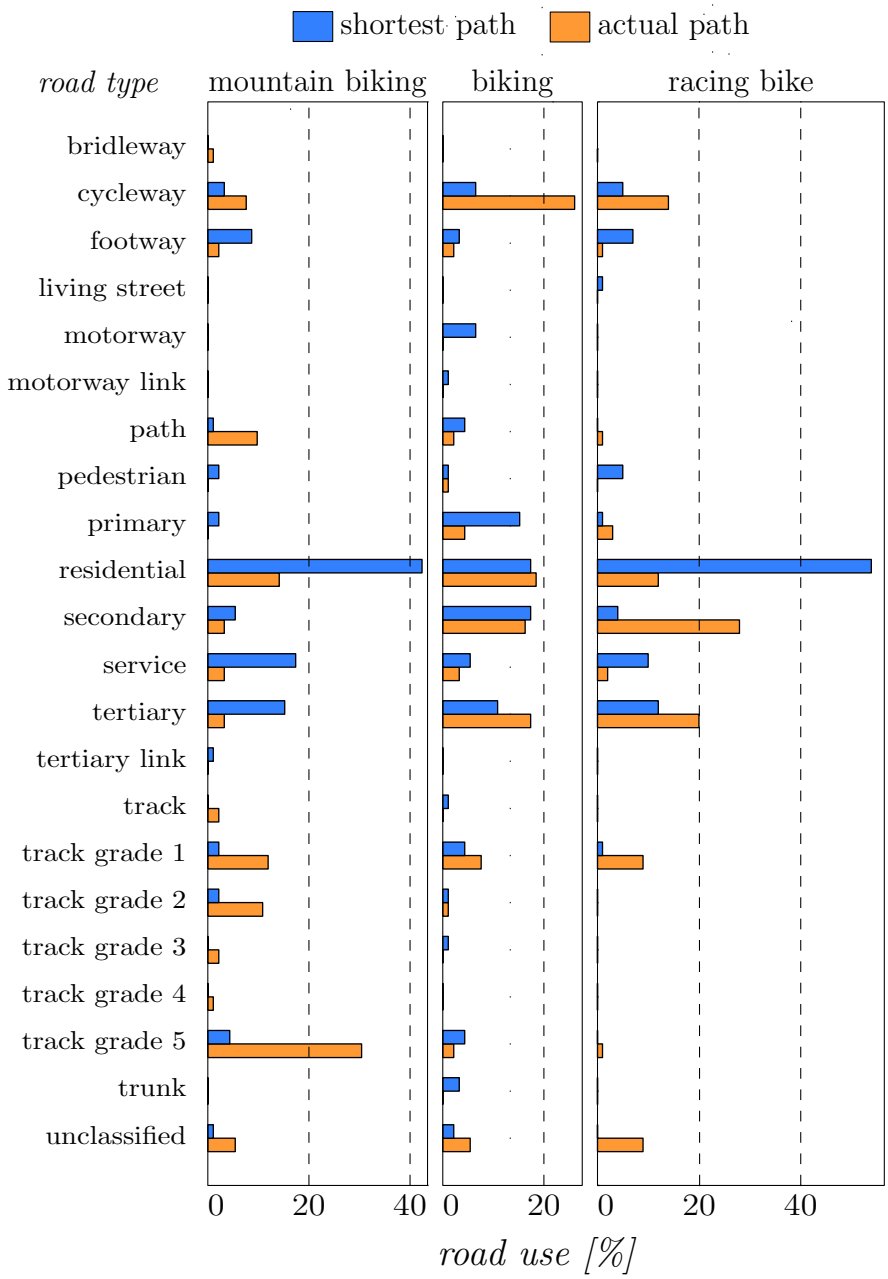


Figure 6.3: Share of different road types among the total length of the trajectories (orange) and the shortest paths connecting the same end vertices (blue), for each of the three types of bicyclists.

road type `track grade 5`, which represents unpaved trails, has the largest share among the paths used by mountain bikers. In contrast, `secondary` is the road type with the largest share among the paths of racing bikers. For other cyclists, `cycleway` is the road type with the highest usage. These observations can be inferred from the large sizes of the corresponding orange bars in Fig. 6.3.

To understand the importance of the blue bars in Fig. 6.3, which represent the share of a road type among the shortest paths, let us discuss the usage of road type `residential` by mountain bikers. The corresponding orange bar is relatively large (actually it comes second after the bar for `track grade 5`) which indicates that mountain bikers quite often use residential streets. However, the corresponding blue bar is much larger than the orange one, which means that if mountain bikers would plan their routes simply based on the routes' geometric lengths, they would end up with an extremely high usage of residential streets. Therefore, we argue that it is legitimate to say that mountain bikers disfavor residential streets. Similarly, based on Fig. 6.3, the following conclusions are most obvious:

- All groups of bicyclists disfavor footways and service streets.
- All groups of bicyclists favor cycleways and streets of type `track grade 1`.
- Mountain bikers additionally prefer paths as well as the types `track grade 2` to `5`, but they disfavor residential streets and tertiary streets.
- Racing bikers favor secondary as well as tertiary streets but disfavor residential streets.
- Other cyclists favor tertiary streets but disfavor primary streets.

We note that statistical tests of significance would be necessary to make more profound statements concerning preferred road types. However, to obtain a binary classification of the road types for the subsequent steps of our analysis, it is most reasonable to apply Equations (6.3) and (6.4). This means, for example, that we say that users of the group `biking` favor residential streets even though the share of residential streets among their routes is only slightly larger than among the corresponding shortest paths (i.e., the blue bar and the orange bar have almost the same size).

6.4.4 Results of the Weight Inference

As a final step, we applied the algorithm of Oehrlein et al. [ONH17] to infer a weighting for each user group[5]. As a result we receive for every given trajectory the size of the segmentation for every $\alpha \in [0, 1]$, in particular the size of a minimal segmentation.

Before analyzing the overall outcome of this step, we would like to take a closer look at the result for a single trajectory (see Fig. 6.4). This trajectory has a total length of 51 km of which 71% are found on roads of favored types. It is a nice example for a mountain bike trajectory in a rather densely populated area: In general, the bicyclist avoided villages and rode through the countryside. Accordingly, the results in Fig. 6.5 approve our classification. The number of milestones that are needed for the segmentation is minimal for $\alpha \in [0.38, 0.43]$. Such a value for α means that this bicyclist accepted detours which are up to 63% longer

[5] see Section 6.A

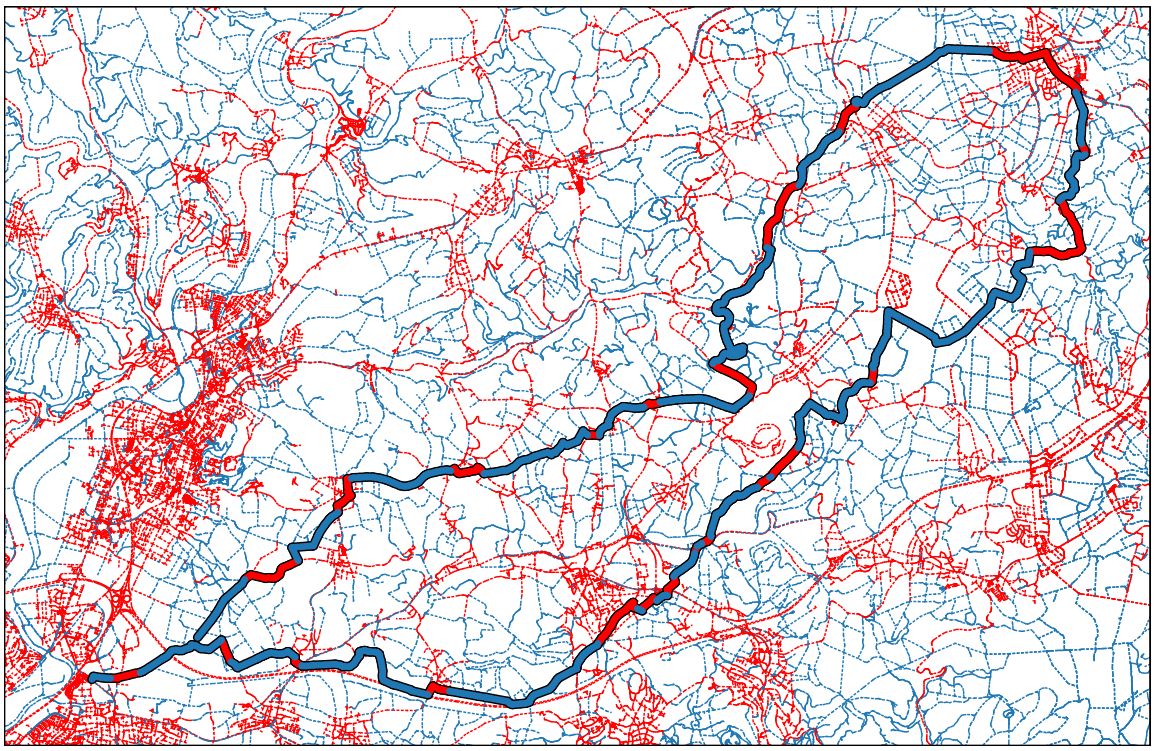


Figure 6.4: A mountain bike trajectory (bold) close to Bonn. Road segments of favored types are depicted as blue lines, those of unfavored types as red lines.

than the shortest path in order to use favored road types instead of unfavored ones. An example explaining this result in more detail can be found in Fig. 6.6.

Thus, any trajectory that has an optimal segmentation for $\alpha < 0.5$ approves our classification. Fig. 6.7 gives an insight how applicable our classification is. In particular, for the user group `racingbiking` four out of five trajectories have optimal segmentations for $\alpha < 0.5$ but not for $\alpha > 0.5$. The weakest classification is the one for the group `mountainbiking`. But even here almost 60% of the trajectories have a minimal segmentation certifying our findings. This group also has the highest proportion of trajectories that have a minimum segmentation for $\alpha > 0.5$ but not for $\alpha < 0.5$ (roughly 10%).

For further analysis, we take the size of a minimal segmentation of a trajectory as 100% and consider for every alpha the necessary number of milestones relative to the size of a minimal segmentation in percent, see the gray lines in Fig. 6.5. Finally, we compute the average percentage of necessary milestones per α for every user group. Figure 6.8 gives an overview of these numbers. At first glance, the results are in accord with the results of Fig. 6.7 and approve our classification. On average, focusing on favored road types is more convenient for every user group than focusing on unfavored road types. Even for the lowest curve, referring to the user group of mountain bikers, it takes more than 50% of milestones extra for $\alpha = 0.9$ in comparison to $\alpha = 0.1$. Taking a closer look, one notices that, for the group `racingbiking`, the best results are obtained for $\alpha \approx 0.485$. That means, that racing bikers are willing to make detours of more than 6% in order to use road types that we have

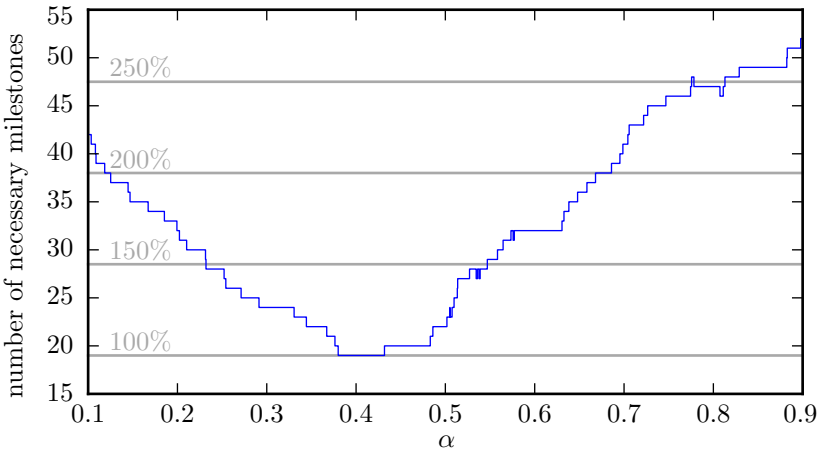


Figure 6.5: Analysis of the milestone segmentation of the trajectory given in Fig. 6.4.

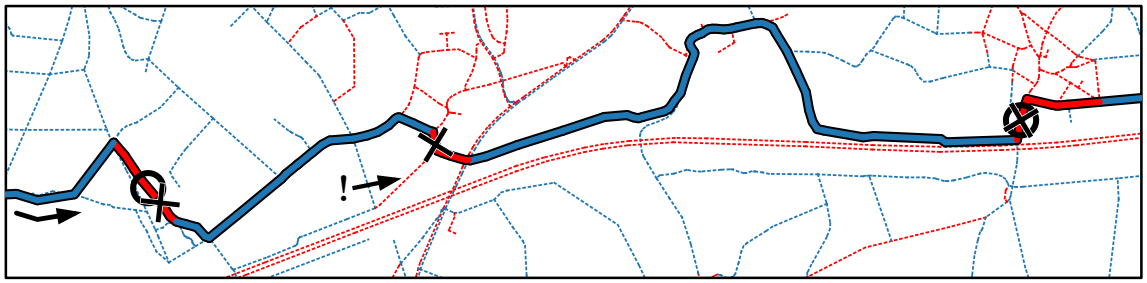


Figure 6.6: Excerpt of the trajectory of Fig. 6.4. Road segments of favored type are colored blue, those of unfavored type are colored red. For this subpath, a segmentation with $\alpha = 0.5$ requires three milestones ($\times$) while already two milestones are sufficient with $\alpha = 0.38$ ($\circ$). The road segment marked with "!" causes an extra milestone for every segmentation with $\alpha \geq 0.48$. Note that this implies that the subpath between the two circles is an optimal path for $\alpha = 0.38$ but not for $\alpha = 0.5$.

recognized as favored ones. But, for `biking` and `mountainbiking` the number of necessary milestones is, on average, minimal for $\alpha = 0.5$. That means, despite an (in parts clear) classification into favored and unfavored road types, the routing results that are best for all users within one of the two groups are achieved when ignoring the classification and simply considering distance. In other words, there is no value for α other than 0.5 that would be better for the whole group – this suggests that one should probably focus on training the parameter α for smaller groups or even for individual users.

6.5 Conclusion

We have presented a novel approach for the classification of bicycle trajectories from crowd-sourced data into different groups. For each group (e.g., mountain bikers) we have identified favored and unfavored road types. Based on this information, we have defined a bicriteria

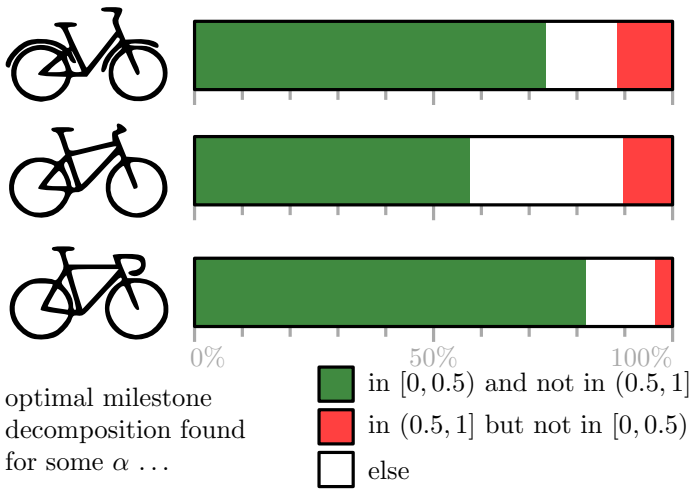


Figure 6.7: Overview of the distribution of minimal segmentations for `cycling` (top), `mountainbiking` (center), and `racingbiking` (bottom). The green bar indicates the share of trajectories that have minimal segmentations only for α values less than or equal to 0.5; the red bar represents the trajectories with minimal segmentations only for α values greater than or equal to 0.5. Trajectories with an optimal segmentation only for $\alpha = 0.5$ as well as all remaining trajectories are represented by the white bar.

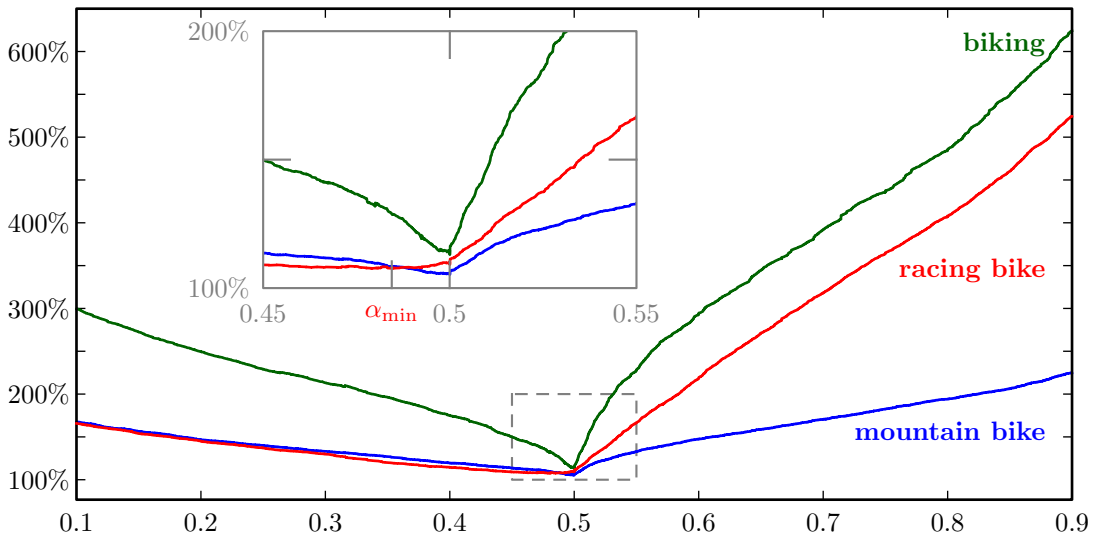


Figure 6.8: Number of milestones as a function of α, summed over all trajectories of the same type and measured in percent relative to the minimum number of milestones. The minimum is attained close to $\alpha = 0.485$ for racing bike and at $\alpha = 0.5$ for biking and mountain biking.

routing model, which assumes that bicyclists favor short routes but on the other hand try to avoid unfavorable road types. We have shown how the trade-off parameter of the model can be learned from the trajectories such that a single edge weighting is obtained that can be used to compute new routes for any two vertices in the cycle network. To this aim, a multi-source data analysis consisting of three steps has been performed.

In the first step, a map-matching approach has been applied in order to combine data from different sources and to extract a significant feature set for the classification of different cyclist groups in an unsupervised manner. We are discriminating between three user groups: `mountainbiking`, `racingbiking` and `biking`. Our results confirmed the user-specified groups with a consensus in 67% of all trajectories. A feature importance analysis revealed that parameters such as the route type (such as circular or simple track), the altitude range, and the difference in length between the trajectory and the respective shortest path turn out to be of great interest for a group categorization.

In the second step, we have identified favored and unfavored road types with regard to each of the three groups. While some types such as `cycleway` are preferred by all groups of cyclists, the analysis also revealed large differences among the different groups. For example, streets of type `tertiary` are clearly favored by the groups `racingbiking` and `biking` but clearly disfavored by the group `mountainbiking`.

In the third step, despite the sparseness of the underlying trajectory sets, we were able to learn a mapping of edge types and edge lengths to edge weights. The results we obtained prove that our approach goes in the right direction. Basically, our classification is proper but needs additional fine-tuning in order to outweigh bicyclists' demand for shortest paths. Particularly for the group `racingbiking` we succeeded and identified a mapping to edge weights that results in paths that are optimal although being 6% longer than shortest paths. For the groups `mountainbiking` and `biking` it turned out that, if the aim is to satisfy all users equally well, the best solution to the routing problem would be simply to minimize the geometric length of the path. Therefore, as a direction for future research, we suggest considering a classification of users into more than three groups or learning the trade-off parameter of the routing model individually for each user. Clustering algorithms such as spectral clustering [NJW02] or mean shift algorithm [CM02] state promising alternatives to k-means, and could facilitate an appropriate choice of the number of clusters. Since we have observed that the users sometimes change their routing criteria even within single trajectories (e.g., since a mountain biker behaves like a normal biker when riding to or back from a hilly region of interest) it may also be reasonable to ask for a partition of a given trajectory into parts that are homogeneous with respect to the routing criteria applied.

Appendix

6.A Appendix: Aggregation of routing criteria

In this section, we elaborate an algorithm for aggregating routing criteria that, so far, has been published as a sketch only [ONH17]. After publishing this outline, we discovered a way of speeding up a subroutine of our algorithm, which also led to an improved overall running time. This speed-up is based on an observation by Funke and Storandt [FS13]. This section for the first time presents the improved version of our algorithm, which we have developed with support from Axel Forsch.

Modern route-planning tools focus on finding optimal paths with respect to, for example, distance, time-, or fuel-consumption. However, various other criteria are of interest, too, like the number of traffic lights along the path or, particularly with respect to bicyclists, the ascent. For multi-criteria routing problems, *Pareto optimal* paths are often considered [KRS10, War87]. A path is considered to be Pareto optimal if an improvement with respect to one criterion can only be achieved by worsening the path's quality with respect to another criterion. Unfortunately, computing Pareto optimal paths is **NP**-hard [War87, GJ90]. A simpler way of dealing with multiple criteria is to linearly combine the given criteria to a new one and apply established single-criterion shortest path algorithms like Dijkstra's Algorithm. For this purpose, however, it is necessary to identify reasonable weighting factors.

Computing such weighting factors is where our algorithm comes in. Gotsman and Kanza [GK13] as well as Lerin et al. [LYT13] have shown independently how to compute a compact representation of a user's path by partitioning it into a minimum number of paths that are optimal with respect to a prescribed criterion. The authors have noted that the better that optimization criterion matches the user's routing preferences, the more compact representations are obtained. Reversing this argument forms the core of our idea: If a routing model induces a partition of a path into a small number of optimal subpaths, it reflects the user's preferences well. In the long run, we plan to design an algorithm that computes weighting factors such that the linear combination of given criteria yields a minimum partition of the path considered. For the moment, it is capable of dealing with two criteria.

In the following, we present an algorithm that, given a path and two criteria, produces an interval of weighting factors yielding linear combinations of criteria such that the partition of the path into optimal subpaths is minimal. The algorithm is composed of two subroutines which are presented in the following subsections. First, an algorithm is designed and presented that determines the possibly empty interval of weighting factors that correspond to a linear combination of criteria for which the given path is optimal. Then, in the following algorithm, this subroutine is called for subpaths of a given path in order to set up a data structure which allows us to determine a minimum segmentation easily.

6.A.1 Determining weighting factors corresponding to path optimality

We consider a (directed) graph $G = (V, E)$ with two distinguished vertices $s, t \in V$ and integral, non-negative edge weights $w_0, w_1 \colon E \to \mathbb{N}_0$. Combining these weights linearly, we obtain an edge weight $w_\alpha = (1 - \alpha) \cdot w_0 + \alpha \cdot w_1$ depending on a weighting factor α. In this section, we describe an algorithm which allows us to identify the *optimality range* of a given simple s-t path P, i.e., the subset $\mathcal{I}_{\text{opt}} \subseteq [0, 1]$ such that P is an optimal path with respect to w_α for every $\alpha \in \mathcal{I}_{\text{opt}}$, see OptimalityRange.

	OptimalityRange
Instance:	A graph $G = (V, E)$, two vertices $s, t \in V$, edge weights $w_0, w_1 \colon E \to \mathbb{N}_0$, an s-t path P.
Question:	Is there a non-empty interval $\mathcal{I}_{\text{opt}} \subseteq [0, 1]$ such that P represents an optimal s-t path with respect to w_α for every $\alpha \in \mathcal{I}_{\text{opt}}$?

According to Section 2.2.1, the weight of a path P is defined as the sum of the weights of its edges, which means

$$\begin{aligned} w_\alpha(P) &= \sum_{e \in P} w_\alpha(e) \\ &= (1 - \alpha) \cdot \sum_{e \in P} w_0(e) + \alpha \cdot \sum_{e \in P} w_1(e) \\ &= (1 - \alpha) \cdot w_0(P) + \alpha \cdot w_1(P) \\ &= w_0(P) + \alpha \cdot \big(w_1(P) - w_0(P)\big). \end{aligned} \tag{6.7}$$

Thus, for a path P, its combined weight $w_\alpha(P)$ describes a line with vertical intercept $w_0(P)$ and slope $w_1(P) - w_0(P)$. In the following, we identify a path with its corresponding line. Consequently, we can consider the set of all s-t paths as a family of lines, see Figure 6.A.1. For a fixed α, we say an s-t path P is *α-optimal* if it is an optimal s-t path with respect to w_α, i.e., its combined weight $w_\alpha(P)$ is minimal among all s-t paths. The set of all paths that are α-optimal for some $\alpha \in [0, 1]$ forms the lower envelope $\mathcal{E} \colon [0, 1] \to \mathbb{R}_{\geq 0}$ of the corresponding line arrangement, see the orange highlighting in Figure 6.A.1(b). The mapping $\mathcal{E}$ can be evaluated for every $\alpha \in [0, 1]$; in our case by computing the weight of a path that is optimal with respect to w_α. The optimality range of an s-t path P can be identified with the subset of $[0, 1]$ in which the line $w_\alpha(P)$ is part of the lower envelope. Thus, the optimality range is either empty, a single value, or an interval. In Figure 6.A.1, the optimality range of the red path is $\emptyset$, the optimality ranges of the green, blue, and black paths are $[0, 1/3]$, $[1/3, 2/3]$, and $[2/3, 1]$, respectively.

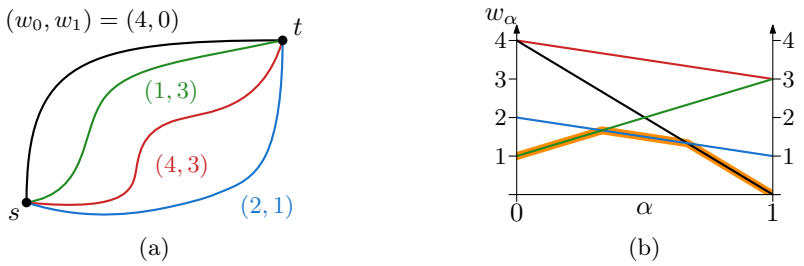


Figure 6.A.1: (a) The set of s-t paths with weights (w_0, w_1) and (b) the corresponding line arrangement; colors are chosen accordingly. In Figure (b), the lower envelope is highlighted in orange.

An existing algorithm for finding elements on the lower envelope Funke and Storandt [FS13] present an algorithm that finds a point on the lower envelope that is also part of the line corresponding to a given path P. In the original scenario, this point corresponds to an α such that P is α-optimal.

The algorithm of Funke and Storandt searches for such a point iteratively. In every step, an interval $\mathcal{I}$ is considered that is guaranteed to contain the optimality range $\mathcal{I}_{\text{opt}}$ completely. From step to step, this interval $\mathcal{I}$ is reduced until a point within the optimality range is found or its size falls below the minimum size of an optimality range, which implies that the optimality range is empty. This minimal size of $\mathcal{I}_{\text{opt}}$ depends solely on the edge weights of the considered graph and can be computed in advance. For this purpose, we introduce $M := \max_{e \in E} \max\{w_0(e), w_1(e)\}$, the maximum occurring weight (w_0 or w_1) of an edge. M exists since we consider a graph with a finite edge set.

Lemma 6.1. *The optimality range $\mathcal{I}_{opt}$ of an s-t path P has a minimum size depending on M. In particular, $\frac{1}{|\mathcal{I}_{opt}|} \in \mathcal{O}(M^2 n^2)$ holds where $n = |V|$.*

Proof. Let $\mathcal{I}_{\text{opt}} = [\alpha', \alpha^*]$ with $0 < \alpha' < \alpha^* < 1$ be the optimality range of P. Then, this interval is defined by two other s-t paths P' and P^*. The lines P and P' intersect at α', the lines P and P^* at α^*. With respect to w_0, the paths P' and P^* differ k' and k^*, respectively, from P and l' and l^* with respect to w_1, see Figure 6.A.2.

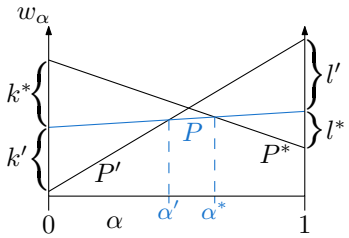


Figure 6.A.2: Example of a path P with an optimality range $[\alpha', \alpha^*]$ and the Paths P' and P^* bounding it.

According to Equation 6.7, the intersection of P and P', i.e. α', can be obtained as follows.

$$\begin{aligned}
& & w_{\alpha'}(P') &= w_{\alpha'}(P) \\
\Leftrightarrow & & w_0(P') + \alpha' \cdot \big(w_1(P') - w_0(P')\big) &= w_0(P) + \alpha' \cdot \big(w_1(P) - w_0(P)\big) \\
\Leftrightarrow & w_0(P) - k' + \alpha' \cdot \big(w_1(P) + l' - w_0(P) + k'\big) &= w_0(P) + \alpha' \cdot \big(w_1(P) - w_0(P)\big) \\
\Leftrightarrow & & -k' + \alpha' \cdot (l' + k') &= 0 \\
\Leftrightarrow & & \alpha' &= \frac{k'}{k' + l'}
\end{aligned}$$

Likewise, $\alpha^* = \frac{k^*}{k^* + l^*}$ can be obtained. Consequently, the size of $\mathcal{I}_{\text{opt}}$ is given as

$$\begin{aligned}
|\mathcal{I}_{\text{opt}}| &= \alpha^* - \alpha' \\
&= \frac{k^*}{k^* + l^*} - \frac{k'}{k' + l'} \\
&= \frac{k^* \cdot (k' + l') - k' \cdot (k^* + l^*)}{(k^* + l^*) \cdot (k' + l')} \\
&= \frac{k^* k' + k^* l' - k' k^* - k' l^*}{(k^* + l^*) \cdot (k' + l')} \\
&= \frac{k^* l' - k' l^*}{(k^* + l^*) \cdot (k' + l')}.
\end{aligned}$$

With the maximum edge weight M, we can bound this interval size from below. Since we consider simple paths only, this gives us a bound of $n \cdot M$ for the maximum weight of a simple path in $G = (V, E)$ where $n = |V|$. Thus, $\max\{k^*, l^*, k', l'\} \leq M \cdot n$. Then, the following holds for the size of the interval $\mathcal{I}_{\text{opt}}$.

$$|\mathcal{I}_{\text{opt}}| \geq \frac{1}{(k^* + l^*) \cdot (k' + l')} \geq \frac{1}{(2 \cdot \max\{k^*, l^*, k', l'\})^2} \geq \frac{1}{4M^2 n^2}. \tag{6.8}$$

In particular, we notice $\frac{1}{|\mathcal{I}_{\text{opt}}|} \leq 4M^2 n^2 \in \mathcal{O}(M^2 n^2)$. The cases $\alpha' = 0$ or $\alpha^* = 1$ lead similarly to a lower bound of $\frac{1}{2Mn}$ which exceeds the bound of the cases considered above.

□

Algorithm 15 is an adaptation the algorithm *Witness Search* by Funke and Storandt [FS13]. Due to the algorithm's importance for this chapter, we elaborate their findings in more detail in the following.

Lemma 6.2. *In every iteration of the algorithm, $\mathcal{I}_{opt} \subseteq \mathcal{I}$ holds.*

Proof. Assume that in the beginning of each iteration step, two values *low* and *upp* defining an interval $\mathcal{I} = [low, upp]$ are given such that $\mathcal{I}_{\text{opt}} \subseteq \mathcal{I}$. This is the case for the first iteration where $\mathcal{I} = [0, 1]$ and, thus, $\emptyset \subseteq \mathcal{I}_{\text{opt}} \subseteq \mathcal{I}$ holds. The minimal value $\mathcal{E}(\alpha)$ of the line arrangement is computed at the central value $\alpha = {}^{(low+upp)}/_2$ of $\mathcal{I}$.

If $\mathcal{E}(\alpha)$ coincides with $w_\alpha(P)$, the optimality range of P contains α and the algorithm terminates. This positive outcome is reached in Line 8.

Algorithm 15: FindOptimalRangeElement

Data: Graph $G = (V, E)$, weights $w_0, w_1 \colon E \to \mathbb{N}_0$, distinguished vertices $s, t \in V$, s-t path P

Result: α such that P is optimal with respect to w_α, possibly NIL if $\mathcal{I}_{\text{opt}} = \emptyset$

$low \leftarrow 0$;
$upp \leftarrow 1$;
while `true` **do**
  $\bar{\alpha} \leftarrow (upp + low)/2$;
  $\bar{P} \leftarrow \operatorname{argmin}_{P'} w_{\bar{\alpha}}(P')$;
  if $w_{\bar{\alpha}}(P) = w_{\bar{\alpha}}(\bar{P})$ **then**
    // P is optimal for $\bar{\alpha}$, i.e., $\bar{\alpha} \in \mathcal{I}_{\text{opt}}$
    return $\bar{\alpha}$;
  if $w_0(P) > w_0(\bar{P}) \wedge w_1(P) > w_1(\bar{P})$ **then**
    // $\bar{P}$ dominates P in $[0, 1]$; thus, $\mathcal{I}_{\text{opt}} = \emptyset$
    return NIL;
  $\bar{\alpha} \leftarrow \dfrac{w_0(\bar{P}) - w_0(P)}{w_1(P) - w_0(P) - w_1(\bar{P}) + w_0(\bar{P})}$;
  if $\bar{\alpha} \notin [low, upp]$ **then**
    // $\bar{P}$ dominates P in $\mathcal{I} = [low, upp]$; thus, $\mathcal{I}_{\text{opt}} = \emptyset$
    return NIL;
  if $w_0(\bar{P}) < w_0(P)$ **then**
    $low \leftarrow \bar{\alpha}$;
  else
    $upp \leftarrow \bar{\alpha}$;

Otherwise, a line Q on the lower envelope is found.

If Q dominates P in all of $\mathcal{I}$, the path P is not optimal for any $\alpha \in [0,1]$ and $\mathcal{I}_{\text{opt}} = \emptyset \subseteq \mathcal{I}$ holds. In this case, Algorithm 15 returns NIL in Line 11. Otherwise, there is an intersection of P and Q within $[0,1]$. If this intersection is found outside $\mathcal{I}$ (see Line 13), the algorithm terminates returning NIL as well.

In any other case, the lines Q and P intersect at some $\bar{\alpha} \in \mathcal{I}$ and $w_\alpha(Q) \leq w_\alpha(P)$ holds for either $\alpha \in [0, \bar{\alpha}]$ or $\alpha \in [\bar{\alpha}, 1]$. This implies $\mathcal{I}_{\text{opt}} \cap [0, \bar{\alpha}[= \emptyset$ or $\mathcal{I}_{\text{opt}} \cap]\bar{\alpha}, 1] = \emptyset$, respectively. Since the respective interval can be excluded from the search interval, $\mathcal{I}$ is reduced to $\mathcal{I} \cap [\bar{\alpha}, 1] = [\bar{\alpha}, upp]$ or $\mathcal{I} \cap [0, \bar{\alpha}] = [low, \bar{\alpha}]$, respectively, and the procedure is repeated for this new search interval $\mathcal{I} \supseteq \mathcal{I}_{\text{opt}}$ in the following iteration.

□

Theorem 6.1. *Algorithm 15 terminates and it takes $\mathcal{O}(\log(Mn))$ iterations.*

Proof. In every step, the interval is limited to a subset of either the left or the right half of $\mathcal{I}$: In Lines 12–19, the search interval $\mathcal{I} = [low, upp]$ gets restricted to $[\bar{\alpha}, upp]$ if $\bar{\alpha} \in [\frac{low+upp}{2}, upp]$ or to $[low, \bar{\alpha}]$ if $\bar{\alpha} \in [low, \frac{low+upp}{2}]$. Hence, the size of $\mathcal{I}$ is reduced by a factor of at least 2 in each iteration. Thus, if the algorithm does not terminate early, the size of $\mathcal{I}$ falls below the minimum interval size at some point. Since the algorithm has not terminated early, P is dominated for both *low* and *upp*. Then one of the following two cases holds:

(i) P is dominated at *low* and *upp* by a single path P', see Figure 6.A.3(e) or (f).

In this case, P is either dominated by P' in all of $[0,1]$ and the algorithm terminates in Line 11 or an intersection $\bar{\alpha} \in [0,1] \setminus \mathcal{I}$ exists. In the latter case, the algorithm terminates in Line 15. In any case, $\mathcal{I}_{\text{opt}} = \emptyset$ holds.

(ii) P is dominated at *low* by P' and at *upp* by P^*, see Figure 6.A.3(a) or (d).

In this case, P is either dominated in $\mathcal{I}$ or optimal for the intersection of P' and P^*. Let us assume without loss of generality that P' is optimal at $\frac{low+upp}{2}$. Then, in Line 12, $\bar{\alpha}$ at the intersection of P and P' gets computed and the search interval is bounded by $[\bar{\alpha}, upp]$. If P is optimal at $\bar{\alpha}$, see Figure 6.A.3(b) and (c), the upper bound *upp* gets $\bar{\alpha}$ as well in the next iteration and the algorithm terminates positively in Line 8. In any other case, P^* dominates P in the search interval and case (i) holds, see Figure 6.A.3(e).

According to Lemma 6.1, the minimum interval size is reached after $\mathcal{O}(\log(M^2n^2)) = \mathcal{O}(\log(Mn))$ steps. □

Theorem 6.2. *The running time of Algorithm 15 is in $\mathcal{O}(\text{SPQ} \cdot \log(Mn))$ where* SPQ *is the running time of a shortest-path query in G.*

Proof. Algorithm 15 consists mainly of a fixed number of basic operations totaling $\mathcal{O}(1)$ time. Solely the operation in Line 5, where an optimal path with respect to w_α is sought, is more expensive and thus dominates the running time of a single iteration of the while loop with $\mathcal{O}(\text{SPQ})$. With regard to Theorem 6.1, the overall run time is in $\mathcal{O}(\text{SPQ} \cdot \log(Mn))$. □

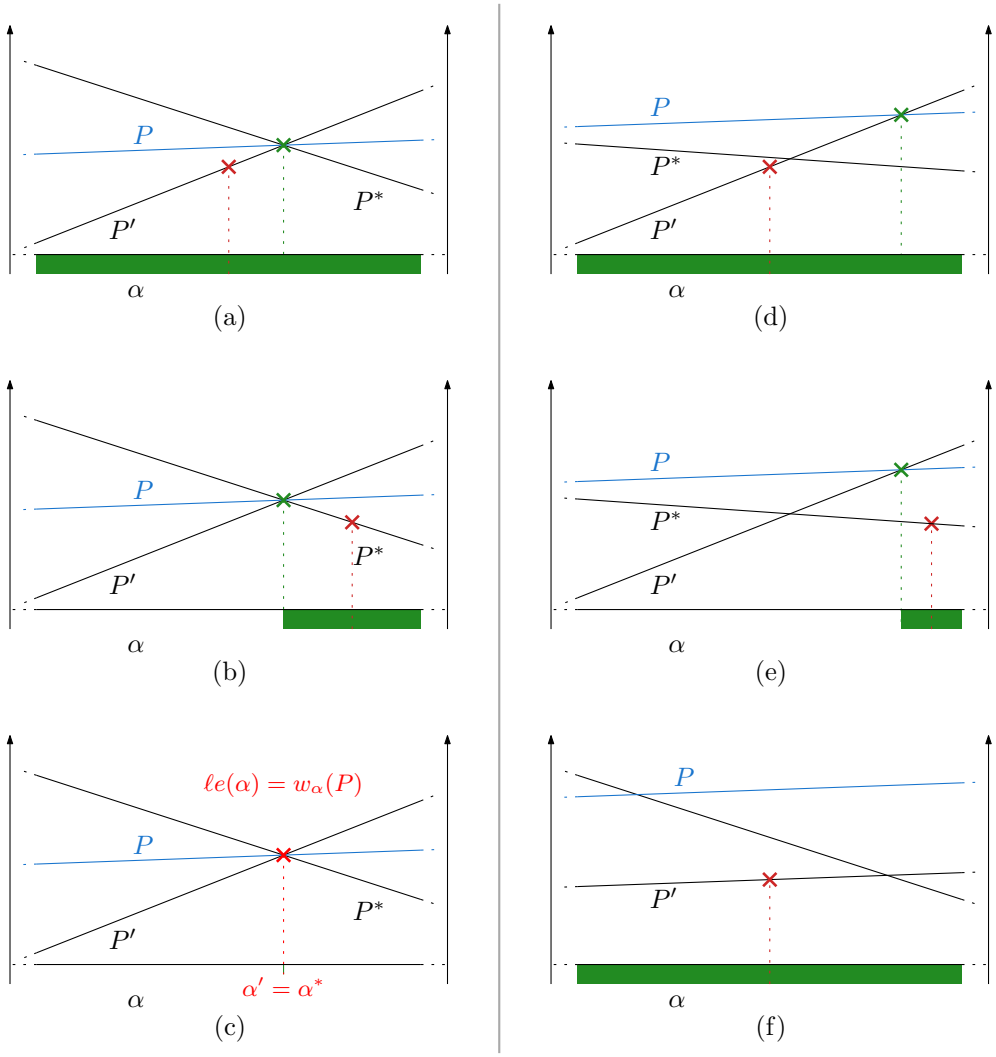


Figure 6.A.3: Final steps of Algorithm 15 for finding an element on both the lower envelope and the line (blue) corresponding to the given path P. Here, the situation is depicted for the case that the algorithm has not terminated early and the size of the search interval $\mathcal{I}$ (green) has fallen below the minimum size derived from the given graph G. The central value α (see Line 4) is marked red, $\bar{\alpha}$ (see Line 12), corresponding to the next reduction of $\mathcal{I}$, is marked green. Regarding the proof of Theorem 6.1, (a)–(c) correspond to the positive result described in case (ii); (d)–(f) correspond to negative results. Figure (f) depicts the special case that a path P' with the same slope dominates P. This is a sufficient yet not necessary condition for a termination of Algorithm 15 in Line 11.

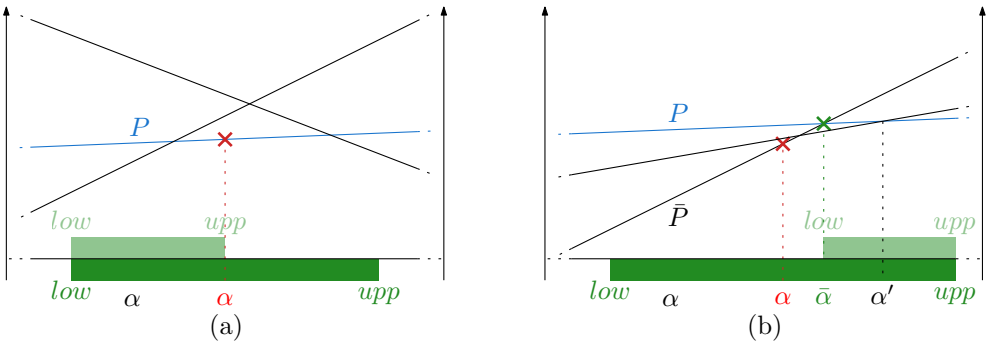


Figure 6.A.4: The search for the lower bound of the optimality range. The search interval is marked green; its current state darker than the next state. (a) If P is optimal with respect to w_0, the search for the lower bound continues in $[low, \alpha]$. (b) At α, the path P is dominated by $\bar{P}$. The search continues in $[\bar{\alpha}, upp]$ where $\bar{\alpha}$ marks the intersection of P and $\bar{P}$.

An extension yielding the optimality range So far, only one Element α within the optimality range $\mathcal{I}_{\text{opt}} = [\alpha', \alpha^*]$ of P is known. Starting with the result of Algorithm 15, the search for the boundary of $\mathcal{I}_{\text{opt}}$ continues. Due to the loop invariant, see Lemma 6.2, we know $\alpha', \alpha^* \in \mathcal{I} = [low, upp]$. In particular, $\alpha' \in [low, \alpha]$ and $\alpha^* \in [\alpha, upp]$ hold. In the following, we describe how to determine α'. The search for α^* is organized symmetrically.

Due to the loop invariant of Algorithm 15, the optimality range lies within the final search interval, i.e. $\mathcal{I}_{\text{opt}} \subseteq \mathcal{I} = [low, upp]$ holds at the beginning of the search for α'. Consequently, if P is an optimal path with respect to w_{low}, the left bounds coincide, $\alpha' = low$ holds. Otherwise, we continue our search similar to Algorithm 15 with a binary search within $\mathcal{I}$. We start our search from $\alpha = \frac{low+upp}{2}$, of which we know that P is optimal with respect to w_α. Then, α' is sought within $[low, \alpha]$ with a binary search that, like Algorithm 15, additionally uses the structure of the line arrangement. Thus, α' is found at the latest when the size of the search interval $\mathcal{I}$ falls below the minimum size of the optimality range.

In every iteration, the search interval $\mathcal{I}$ is reduced by at least half, see Figure 6.A.4. We consider the central value α of $\mathcal{I}$. If P is optimal with respect to w_α, the search continues in $[low, \alpha]$, see Figure 6.A.4(a). Otherwise, there exists another path $\bar{P}$ that is optimal with respect to w_α. This path $\bar{P}$ has the same weight as P for some value $\bar{\alpha}$. Since $\bar{P}$ dominates P in $[0, \bar{\alpha}]$, the lower bound of the optimality range is found if P is an optimal path with respect to $w_{\bar{\alpha}}$. Otherwise, the search continues in $[\bar{\alpha}, upp]$, see Figure 6.A.4(b). This procedure is summarized in Algorithm 16.

The search for α^*, the upper bound of the optimality range $\mathcal{I}_{\text{opt}} = [\alpha', \alpha^*]$, can be done symmetrically. Hence, in the worst case, the search interval needs to be reduced from $[0, 1]$ to below the minimal size twice; once for the lower bound α', once for the upper bound α^*. Consequently, the optimality range can also be found within $\mathcal{O}(\text{SPQ} \cdot \log(Mn))$ time.

6.A.2 Segmenting a path into a minimum number of optimal subpaths

Being able to solve OptimalityRange, we now deal with the problem of segmenting a path into a minimum number of α-optimal subpaths with $\alpha \in [0, 1]$, see

Algorithm 16: FindOptimalRangeLowerBound

Data: path P, return value α of Algorithm 15, *low* as given after Algorithm 15
Result: lower bound α'

```
1  upp ← α;
2  while true do
3      ᾱ ← (low + upp)/2;
4      P̄ ← argmin_{P'} w_ᾱ(P');
5      if w_ᾱ(P) = w_ᾱ(P̄) then
6          if w_1(P) < w_1(P̄) then
7              return ᾱ;
8          upp ← ᾱ;
9      else
10         ᾱ ← (w_0(P̄) − w_0(P)) / (w_1(P) − w_0(P) − w_1(P̄) + w_0(P̄));
11         P̄ ← argmin_{P'} w_ᾱ(P');
12         if w_ᾱ(P) = w_ᾱ(P̄) then
13             return ᾱ;
14         low ← ᾱ;
```

MilestoneSegmentation below. That means we search for an $\alpha \in [0,1]$ and a segmentation of a Path P into a minimal number h of subpaths $\{P_1, \ldots, P_h\}$ such that every subpath P_i with $i \in \{1, \ldots, h\}$ is α-optimal.

	MilestoneSegmentation
Instance:	A graph $G = (V, E)$,
	edge weights $w_0, w_1 \colon E \to \mathbb{N}_0$,
	a path $P = \langle v_0, \ldots, v_\ell \rangle$ in G,
	an integer $k \in \{1, \ldots, \ell\}$.
Question:	Is there an $\alpha \in [0,1]$ such that a segmentation of P into k or less α-optimal subpaths exists?

For this purpose, we apply a concept known in the relevant literature as *start-stop matrix* [ABB+14, ADvK+15]. Hence, for a path P consisting of k vertices, we consider a $(k \times k)$-matrix $\mathcal{M}$ of sub-intervals of $[0,1]$. The entry $\mathcal{M}[i,j]$ in row i and column j corresponds to the optimality range of the subpath of P starting at its i-th vertex and ending at its j-th vertex. Hence, for an α-optimal path P, the entry of $\mathcal{M}$ describing the complete path contains α, i.e., $\alpha \in \mathcal{M}[1,k]$. Since we are interested only in subpaths with the same orientation as P, we focus on the upper triangle matrix with $i \leq j$ and consider $\mathcal{M}[i,j] = \emptyset$ for $i > j$. Then, given $\alpha \in [0,1]$, finding a segmentation of P into a minimal number of

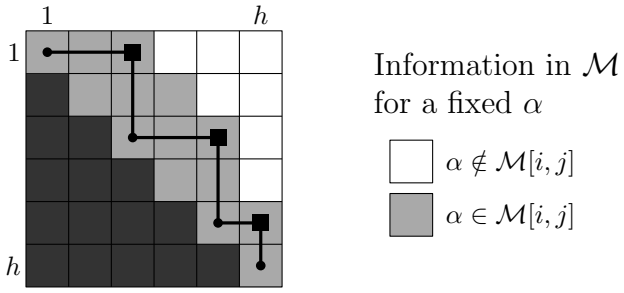


Figure 6.A.5: Depiction of a start-stop matrix corresponding to a path consisting of $h = 6$ edges for a fixed $\alpha \in [0, 1]$. In the upper triangle, white squares indicate that $\alpha \notin \mathcal{M}[i, j]$ whereas gray squares indicate the opposite. The black line represents a resulting minimum segmentation of the path into 3 subpaths with the indices $i_0 = 1, i_1 = 3, i_2 = 5, i_3 = 6$. The decisive intervals in $\mathcal{M}$ are marked with black squares; namely $\mathcal{M}[1, 3]$, $\mathcal{M}[3, 5]$, and $\mathcal{M}[5, 6]$.

α-optimal subpaths corresponds to finding a minimal number of indices $i_0 < i_1 < \ldots < i_h$ with $i_0 = 1$ and $i_h = k$ such that $\alpha \in \mathcal{M}[i_j, i_{j+1}]$ for $0 \leq j < h$; i.e. traversing the matrix $\mathcal{M}$ in a staircase-like manner, see Figure 6.A.5.

Due to *substructure optimality*, subpaths of optimal paths are optimal as well [CLRS09]. Hence, for $i < k < j$, both $\mathcal{M}[i, j] \subseteq \mathcal{M}[i, k]$ and $\mathcal{M}[i, j] \subseteq \mathcal{M}[k, j]$ hold. This results in the structure visible in Figure 6.A.5 where no white cell is on the left side of or below a gray cell for every $\alpha \in [0, 1]$. As a consequence, for a fixed α, it is easy to find a solution to the segmentation problem once the start-stop matrix is set up. According to Buchin et al. [BDvKS11], for example, an exact solution to this problem can be found with a greedy approach in $\mathcal{O}(h)$ time.

Since we consider a finite set of intervals, we know that if a minimal solution exists for an $\alpha \in [0, 1]$, it also exists for one of the values bounding the intervals in $\mathcal{M}$. Consequently, each of the $\mathcal{O}(h^2)$ optimality ranges that need to be computed yields at most two candidates for the solution. For each of these candidates a minimum segmentation needs to be computed in $\mathcal{O}(h)$ time each. Thus, we end up with a total running time of $\mathcal{O}(h^2 \cdot (h + \mathrm{SPQ} \log(Mn))$ where n denotes the number of vertices in the graph and h denotes the number of vertices in the considered path. Thus, the algorithm is efficient and yields an exact solution to MilestoneSegmentation. The solution consists of an interval producing the best fitting aggregated criterion with respect to the input criteria.

7 Conclusion and outlook

In the following, we summarize the results of this thesis. After closing remarks on the results achieved in the previous chapters in Section 7.1, we give an overview of open problems which seem worthwhile to pursue further in Section 7.2.

7.1 Conclusion

In this thesis, we have presented different kinds of aggregation problems and, for their solution, different kinds of exact optimization algorithms. As stated in Chapter 1, there is semantic, geometric and temporal generalization and a clear categorization is often impossible. While the problem dealt with in Chapter 3 focuses on geometric aspects rather than semantic ones, the focus was shifted towards semantic aspects in Chapters 4 and 5. Eventually, in Chapter 6, the aggregation is done in order to simplify the semantic data and the geometric representation plays only a minor role.

The considered problems also differ in complexity and so do the algorithms we developed in order to deal with them. In Chapter 4, we applied sophisticated techniques for solving integer linear programs to a spatial problem. In the field of combinatorial optimization, these methods are established and a common approach for tackling **NP**-complete problems. For the problems described in the remaining Chapters 3, 5, and 6, we were able to develop efficient algorithms.

The efficient algorithms we presented demonstrate that solving aggregation problems exactly is reasonable. This impression is intensified by the fact that two of the ideas presented here stood out at the conferences at which they were presented, either as award-winning contribution or runner-up. Hence, we encourage researchers to search for problem-specific exact algorithmic solutions.

However, in some cases, like the problem presented in Chapter 4, there is no way of avoiding heuristic approaches in order to find a good solution to a problem in an acceptable amount of time. In this case, existing and continuously improved metaheuristic algorithms provide helpful tools for this purpose. Heuristic approaches are of particular interest if the problem proves to be **NP**-hard. Altogether, we consider an interplay between heuristic and exact algorithms as desirable. On the one hand, exact algorithms yield optimal solutions to examined problems, which can be used as benchmarks for faster heuristic approaches. On the other hand, like Puchinger and Raidl suggest [PR05], a combination of exact and heuristic algorithms may increase the quality of both approaches: exact algorithms can be sped-up by intermediate explorations of the solution space and the quality of solutions found with heuristics can be increased, for example, by solving sub problems optimally. These are interesting aspects for future research.

7.2 Outlook

To the problem introduced in Chapter 3, we have presented an algorithm that is both efficient and exact. Based on a computation of a shortest-path tree in advance, which can be done in $\mathcal{O}(n \log n + m)$ time, our algorithm solves TreeSummary in linear time. The original work already contained our proposal to consider a dynamic version of the problem. We already shed some light on the dynamics [OvDH17], but, in the hope of finding an easy adaption of the shortest-path tree, we focused mainly on how a moving source alters the shortest-path tree. For future work, it would be interesting to examine the consequences of replacing the preprocessing step of computing one shortest-path tree with running an all-pair shortest-path algorithm. Subsequently retrieving the shortest-path tree for locations "between" vertices of the road graph is of particular interest, especially with regard to *virtual vertices*. Furthermore, we planned to apply our algorithm to road graphs with a pre-defined partitioning of the road vertices (e.g., into administrative regions). This problem has proven to be more complex than expected. Since these partitions, i.e. administrative regions, are accessible via more than one route, in general, the lowest common ancestor of all vertices within such a region lies outside the region. Consequently, the lowest common ancestor does not seem to be a reasonable choice for representing the region. Moreover, finding a single representation of the region seems impossible with a straight-forward adaptation of our approach. Hence, tackling this problem more open-mindedly than in the past is of interest to us.

The cutting-plane algorithm for aggregating areas presented in Chapter 4 deals with guaranteeing contiguity of aggregated regions. This problem is of major interest [ZS83, CSGW04, DCM11, DVGE18] and, according to our experiments, our model is a huge improvement compared to the current state-of-the-art approach to this problem. Yet, it is of limited practical use as it is only applicable to rather small instances. However, with the application of a cutting-plane algorithm, algorithmic resources are not yet drained. For example, heuristic algorithms can be combined with our exact approach in order to find optimal results faster. Puchinger and Raidl [PR05] suggest, for example, using metaheuristics for obtaining incumbent solutions and bounds in order to support the ILP solver. Puchinger and Raidl also discuss the other way round: using exact algorithm to develop or improve heuristic approaches. With regard to existing heuristics [BEL03, DAR12, HW10a, HWS$^+$65, LCG14, MCVL02] and, in particular, to the problem's complexity, we consider this an interesting approach. Furthermore, we work on an algorithm that solves the problem of area aggregation while considering line features as obstacles. These can be roads, for example. Aggregating areas located on different sides of a motorway might not be desired. On this matter, in cooperation with Sven Gedicke, we decided to work on an heuristic approach instead of an MILP as our ideas for formalizing the cost for aggregating such separated areas cannot be expressed as linear constraints.

In Chapter 5, we present a recent approach to assessing urban green space. Consequently, the outlook section of this chapter gives a good overview of our current ideas for future research. This work originates from a discussion panel with domain experts. Hence, we planned to publish a version of our implementation that is easy to use. The positive feedback we received at the conference where the basic version of this approach was first presented

(GIScience 2018 [NOLH18]) confirmed us in our intention. Hence, this is an important point that completes the ideas we presented in the chapter's outlook section.

As indicated in the introduction to Chapter 6, there is a lot of work to do in the context of inferring routing preferences from trajectories. In particular, publishing the formalization of the general concept, our idea and our framework, with a solid foundation on existing literature is necessary. This is of special interest since, in the meantime, research on this topic has continued. We aim for extending our approach to more than two criteria; first steps have been undertaken in the context of a Master's thesis. Furthermore, we see potential in analyzing a multitude of trajectories and aggregating them into clusters with similar routing preferences. Further research is pursued within a project for *Inferring Personalized Multi-criteria Routing Models from Sparse Sets of Voluntarily Contributed Trajectories* which is part of a priority project of the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) on volunteered geographic information (VGI) [HF19].

Bibliography

[AA10] N. Andrienko and G. Andrienko. Spatial generalization and aggregation of massive movement data. *IEEE Transactions on Visualization and Computer Graphics*, 17(2):205–219, 2010. doi:10.1109/TVCG.2010.44.

[AAS10] T. Ali, S. Asghar, and N. A. Sajid. Critical Analysis of DBSCAN Variations. In *2010 International Conference on Information and Emerging Technologies*, 2010. doi:10.1109/ICIET.2010.5625720.

[ABB+14] S. P. A. Alewijnse, K. Buchin, M. Buchin, A. Kölzsch, H. Kruckenberg, and M. A. Westenberg. A framework for trajectory segmentation by stable criteria. In *Proc. 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '14)*, pages 351–360, 2014. doi:10.1145/2666310.2666415.

[ADvK+15] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of Trajectories on Nonmonotone Criteria. *ACM Transactions on Algorithms*, 12(2):1–28, 2015. doi:10.1145/2660772.

[AEHS03] J. C. J. H. Aerts, E. Eisinger, G. B. M. Heuvelink, and T. 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.

[Alt97] M. Altman. The Computational Complexity of Automated Redistricting: Is Automation the Answer? *Rutgers Computer and Law Technology Journal*, 23(1):81–142, 1997. `http://heinonline.org/HOL/Page?handle=hein.journals/rutcomt23&g_sent=1&id=87&collection=journals`.

[ÁMLM13] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel. The maximum weight connected subgraph problem. In M. Jünger and G. Reinelt, editors, *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, pages 245–270. Springer, Berlin, Germany, 2013. doi:10.1007/978-3-642-38189-8_11.

[And03] K.-H. Anders. A Hierarchical Graph-Clustering Approach to find Groups of Objects. In *Proceedings 5th workshop on progress in automated map generalization*, pages 1–8, 2003.

[AP02] P. K. Agarwal and C. M. Procopiuc. Exact and Approximation Algorithms for Clustering. *Algorithmica*, 33(2):201–226, 2002. doi:10.1007/s00453-001-0110-y.

[AS01] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *Proc. 28th Conference on Compututer Graphics and interactive techniques*, pages 241–249, 2001. doi:10.1145/383259.383286.

[AV07] D. Arthur and S. Vassilvitskii. k-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007*, pages 1027–1035, 2007. doi:10.1145/1283383.1283494.

[BBW05] M. Bader, M. Barrault, and R. Weibel. Building displacement over a ductile truss. *International Journal of Geographical Information Science*, 19(8-9):915–936, 2005. doi:10.1080/13658810500161237.

[BDG12] J. Broach, J. Dill, and J. Gliebe. Where do cyclists ride? A route choice model developed with revealed preference GPS data. *Transportation Research Part A: Policy and Practice*, 46(10):1730–1740, 2012.

[BDGK19] K. Buchin, A. Driemel, J. Gudmundsson, and I. Kostitsyna. Approximating (k, l)-center clustering for curves. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2922–2938. Society for Industrial and Applied Mathematics, 2019, arXiv:1805.01547v2. doi:10.1137/1.9781611975482.181.

[BDvKS11] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(3):33–63, 2011. doi:10.5311/JOSIS.2011.3.66.

[BEL03] B. Bozkaya, E. Erkut, and G. Laporte. A tabu search heuristic and adaptive memory procedure for political districting. *European Journal of Operational Research*, 144:12–26, 2003. doi:10.1016/S0377-2217(01)00380-0.

[Ber83] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. The University of Wisconsin Press, 1st edition, 1983.

[BGK+14] E. Brunel, A. Gemsa, M. Krug, I. Rutter, and D. Wagner. Generalizing geometric graphs. *Journal of Graph Algorithms and Applications*, 18(1):35–76, 2014. doi:10.7155/jgaa.00314.

[BJS13] A. Balteanu, G. Jossé, and M. Schubert. Mining driving preferences in multi-cost networks. In *Proc. 13th International Symposium on Advances in Spatial and Temporal Databases (SSTD '13)*, pages 74–91, 2013.

[BL07] R. Blanch and E. Lecolinet. Browsing zoomable treemaps: Structure-aware multi-scale navigation techniques. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1248–1253, 2007.

[BLVN09] T. Baycan-Levent, R. Vreeker, and P. Nijkamp. A Multi-Criteria Evaluation of Green Spaces in European Cities. *European Urban and Regional Studies*, 16(2):193–213, 2009. doi:10.1177/0969776408101683.

[BN68] M. Bellmore and G. L. Nemhauser. The Traveling Salesman Problem: A Survey. *Operations Research*, 16(3):538–558, 1968.

[BO16] C. Bergman and J. Oksanen. Conflation of OpenStreetMap and mobile sports tracking data for automatic bicycle routing. *Transactions in GIS*, 20(6):848–868, 2016.

[BØTS06] T. Bjerke, T. Østdahl, C. Thrane, and E. Strumse. Vegetation density of urban parks and perceived appropriateness for recreation. *Urban Forestry & Urban Greening*, 5:35–44, 2006. doi:10.1016/j.ufug.2006.01.006.

[BRMC05] A. L. Bedimo-Rung, A. J. Mowen, and D. A. Cohen. The significance of parks to physical activity and public health: A conceptual model. *American Journal of Preventive Medicine*, 28(2):159–168, 2005. doi:10.1016/j.amepre.2004.10.024.

[BTA+07] O. Barbosa, J. A. Tratalos, P. R. Armsworth, R. G. Davies, R. A. Fuller, P. Johnson, and K. J. Gaston. Who benefits from access to green space? A case study from Sheffield, UK. *Landscape and Urban Planning*, 83(2-3):187–195, 2007. doi:10.1016/j.landurbplan.2007.04.004.

[BW88] K. E. Brassel and R. Weibel. A review and conceptual framework of automated map generalization. *International Journal of Geographical Information Systems*, 2(3):229–244, 1988. doi:10.1080/02693798808927898.

[BYB+13] M. A. Borkin, C. S. Yeh, M. Boyd, P. Macko, K. Z. Gajos, M. Seltzer, and H. Pfister. Evaluation of filesystem provenance visualization tools. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2476–2485, 2013.

[CBG08] A. Comber, C. Brunsdon, and E. Green. Using a GIS-based network analysis to determine urban greenspace accessibility for different ethnic and religious groups. *Landscape and Urban Planning*, 86(1):103–114, 2008. doi:10.1016/j.landurbplan.2008.01.002.

[CC10] T. J. Cova and R. L. Church. Contiguity Constraints for Single-Region Site Search Problems. *Geographical Analysis*, 32(4):306–329, Sep 2010. doi:10.1111/j.1538-4632.2000.tb00430.x.

[CCG+13] R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. Imposing Connectivity Constraints in Forest Planning Models. *Operations Research*, 61(4):824–836, 2013. doi:10.1287/opre.2013.1183.

[CD00] J. M. Casado-Díaz. Local Labour Market Areas in Spain: A Case Study. *Regional Studies*, 34(9):843–856, 2000. doi:10.1080/00343400020002976.

[Chr76a] N. Christofides. The Vehicle Routing Problem. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 10(1):55–70, 1976.

[Chr76b] N. Christofides. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University*, 1976.

[CJH10] E. Coombes, A. P. Jones, and M. Hillsdon. The relationship of physical activity and overweight to objectively measured green space accessibility and use. *Social Science & Medicine*, 70(6):816–822, 2010. doi:10.1016/j.socscimed.2009.11.020.

[CJW06] J. A. Carlson, A. Jaffe, and A. Wiles. *The millennium prize problems*. American Mathematical Society, 2006.

[CKB09] A. Cockburn, A. Karlson, and B. B. Bederson. A review of overview+detail, zooming, and focus+context interfaces. *ACM Computing Surveys*, 41(1):2:1–2:31, 2009. doi:10.1145/1456650.1456652.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction To Algorithms*. MIT Press, Cambridge, MA, USA, 1990.

[CLRS09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 3rd edition, 2009.

[CM02] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[CMS+07] D. A. Cohen, T. L. McKenzie, A. Sehgal, S. Williamson, D. Golinelli, and N. Lurie. Contribution of public parks to physical activity. *American Journal of Public Health*, 97(3):509–514, 2007. doi:10.2105/AJPH.2005.072447.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[CPL17] CPLEX, IBM ILOG. 12.8 User's Manual, 2017. `https://www.ibm.com/analytics/cplex-optimizer`.

[CSGW04] F. Caro, T. Shirabe, M. Guignard, and A. Weintraub. School Redistricting: Embedding GIS Tools with Integer Programming. *Journal of the Operational Research Society*, 55(8):836–849, 2004. doi:10.1057/palgrave.jors.2601729.

[CvDH14] M. Chimani, T. C. van Dijk, and J.-H. 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 (ACM GIS '14)*, pages 243–252, 2014. doi:10.1145/2666310.2666414.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[DAR12] J. C. Duque, L. Anselin, and S. J. Rey. The Max-p-Regions Problem. *Journal of Regional Science*, 52(3):397–419, Aug 2012. doi:10.1111/j.1467-9787.2011.00743.x.

[dBCvKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, Berlin, Germany, 2008. doi:10.1007/978-3-540-77974-2.

[dBvKS98] M. de Berg, M. van Kreveld, and S. Schirra. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems*, 25(4):243–257, 1998. doi:10.1559/152304098782383007.

[DCM11] J. C. Duque, R. L. Church, and R. S. Middleton. The p-Regions Problem. *Geographical Analysis*, 43(1):104–126, 2011. doi:10.1111/j.1538-4632.2010.00810.x.

[DDS09] C. Dyken, M. Dæhlen, and T. Sevaldrud. Simultaneous curve simplification. *Journal of Geographical Systems*, 11(3):273–289, 2009. doi:10.1007/s10109-009-0078-8.

[DH99] A. Drexl and K. Haase. Fast Approximation Methods for Sales Force Deployment. *Management Science*, 45(10):1307–1323, 1999. doi:10.1287/mnsc.45.10.1307.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[DMM+97] S. Doddi, M. V. Marathe, A. Mirzaian, B. M. E. Moret, and B. Zhu. Map labeling and its generalizations. In M. E. Saks, editor, *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1997*, pages 148–157. ACM/SIAM, 1997. `http://dl.acm.org/citation.cfm?id=314161.314250`.

[DVGE18] J. C. Duque, M. C. Vélez-Gallego, and L. C. Echeverri. On the Performance of the Subtour Elimination Constraints Approach for the p-Regions Problem: A Computational Study. *Geographical Analysis*, 50:32–52, 2018. doi:10.1111/gean.12132.

[DvKS08] J. Damen, M. van Kreveld, and B. Spaan. High Quality Building Generalization by Extending the Morphological Operators. In *11th ICA Workshop on Generalization and Multiple Representation*, pages 1–12, 2008.

[EK72] J. Edmonds and R. M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *Journal of the ACM*, 19(2):248–264, 1972. doi:10.1145/321694.321699.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, volume 34, pages 226–231, 1996. `http://www.aaai.org/Library/KDD/1996/kdd96-037.php`.

[Eur11] European Observation Network for Territorial Development and Cohesion. European regions 2010: Economic welfare and unemployment, 2011. `http://www.espon.eu/main/Menu_Publications/Menu_MapsOfTheMonth/map1103.html`.

[Eur15] Eurostat. Population by single year of age and NUTS 3 region, 2015. `http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=cens_11ag_r3&lang=en`.

[Fab15] A. Fabri. 2D polyline simplification. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.6 edition, 2015. `http://doc.cgal.org/4.6/Manual/packages.html#PkgPolylineSimplification2Summary`.

[FG09] R. A. Fuller and K. J. Gaston. The scaling of green space coverage in European cities. *Biology Letters*, 5(3):352–355, 2009. doi:10.1098/rsbl.2009.0010.

[FJF55] L. R. Ford Jr. and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the hitchcock problem. Technical report, RAND Corp., 1955.

[FJF56a] L. R. Ford Jr. and D. R. Fulkerson. Maximal flow through a network. *Journal canadien de mathématiques*, 8(0):399–404, 1956. doi:10.4153/CJM-1956-045-5.

[FJF56b] L. R. Ford Jr. and D. R. Fulkerson. Solving the transportation problem. *Management Science*, 3(1):24–32, 1956. doi:10.1287/mnsc.3.1.24.

[FLS16] S. Funke, S. Laue, and S. Storandt. Deducing individual driving preferences for user-aware navigation. In *Proc. 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '16)*, pages 14:1–14:9, 2016.

[FM12] M. Fischetti and M. Monaci. Cutting plane versus compact formulations for uncertain (integer) linear programs. *Mathematical Programming Computation*, 4(3):239–273, Sep 2012. doi:10.1007/s12532-012-0039-y.

[For10] S. Fortunato. Community detection in graphs. *Physics Report*, 486(3–5):75–174, 2010, arXiv:0906.0612v2. doi:10.1016/j.physrep.2009.11.002.

[FRCDMB08] F. Flórez-Revuelta, J. M. Casado-Díaz, and L. Martínez-Bernabeu. An evolutionary approach to the delineation of functional areas based on travel-to-work flows. *International Journal of Automation and Computing*, 5(1):10–21, 2008. doi:10.1007/s11633-008-0010-6.

[FS13] S. Funke and S. Storandt. Polynomial-time Construction of Contraction Hierarchies for Multi-criteria Objectives. In *2013 Proceedings of the 15th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 41–54. Society for Industrial and Applied Mathematics, 2013. doi:10.1137/1.9781611972931.4.

[Fur86] G. W. Furnas. Generalized fisheye views. In *Proc. ACM Conference on Human Factors in Computing Systems (SIGCHI'86)*, pages 16–32, 1986.

[Gal03] M. Galanda. *Automated Polygon Generalization in a Multi Agent System.* dissertation, University of Zurich, Switzerland, 2003.

[Gas90] S. I. Gass. On solving the transportation problem. *Journal of the Operational Research Society*, 41(4):291–297, 1990. doi:10.1057/jors.1990.50.

[GCD02] B. Giles-Corti and R. J. Donovan. The relative influence of individual, social and physical environment determinants of physical activity. *Social Science & Medicine*, 54:1793–1812, 2002. doi:10.1016/S0277-9536(01)00150-2.

[GGBS11] R. Guercke, T. Götzelmann, C. Brenner, and M. Sester. Aggregation of LoD 1 building models as an optimization problem. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(2):209–222, 2011. doi:10.1016/j.isprsjprs.2010.10.006.

[GJ90] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., San Francisco, 1990.

[GK13] R. Gotsman and Y. Kanza. Compact representation of GPS trajectories over vectorial road networks. In *Proc. 13th International Conference on Advances in Spatial and Temporal Databases (SSTD '13)*, pages 241–258, Berlin, Germany, 2013. Springer-Verlag. doi:10.1007/978-3-642-40235-7_14.

[GLM+17] A. Grønlund, K. G. Larsen, A. Mathiasen, J. S. Nielsen, S. Schneider, and M. Song. Fast Exact k-Means , k-Medians and Bregman Divergence Clustering in 1D. *arXiv preprint*, pages 1–16, 2017, arXiv:1701.07204v4.

[GN70] R. S. Garfinkel and G. L. Nemhauser. Optimal Political Districting by Implicit Enumeration Techniques. *Management Science*, 16(8):B–495–B–508, 1970. doi:10.1287/mnsc.16.8.B495.

[Gon85] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. doi:10.1016/0304-3975(85)90224-5.

[Goo07] M. F. Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007. doi:10.1007/s10708-007-9111-y.

[GRL+16] K. Gupta, A. Roy, K. Luthra, S. Maithani, and Mahavir. GIS based analysis for assessing the accessibility at hierarchical levels of urban green spaces. *Urban Forestry & Urban Greening*, 18:198–211, 2016. doi:10.1016/j.ufug.2016.06.005.

[GRM+17] K. Grunewald, B. Richter, G. Meinel, H. Herold, and R.-U. Syrbe. Proposal of indicators regarding the provision and accessibility of green spaces for

assessing the ecosystem service "recreation in the city" in Germany. *International Journal of Biodiversity Science, Ecosystem Services & Management*, 13(2):26–39, 2017. doi:10.1080/21513732.2017.1283361.

[GSSD08] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proc. 7th International Workshop on Experimental Algorithms (WEA '08)*, pages 319–333, 2008.

[GSSV12] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

[Gur18] Gurobi Optimization, Inc. Gurobi optimizer 8.1 reference manual, 2018. `https://www.gurobi.com/`.

[Har04] T. Hartig. Restorative Environments. In *Encyclopedia of applied psychology*, volume 3, pages 273–279. Elsevier Inc., 2004. doi:10.1016/B0-12-657410-3/00821-7.

[Hau07] J.-H. Haunert. Optimization methods for area aggregation in land cover maps. In *Proceedings of the 23rd International Cartographic Conference*, 2007.

[Hau08] J.-H. Haunert. *Aggregation in Map Generalization by Combinatorial Optimization*. Dissertation, Leibniz Universität Hannover, Germany, 2008.

[HB12] J.-H. Haunert and B. Budig. An algorithm for map matching given incomplete road data. In *Proc. 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '12)*, pages 510–513, 2012.

[HdVA+17] K. T. Hegetschweiler, S. de Vries, A. Arnberger, S. Bell, M. Brennan, N. Siter, A. Stahl Olafsson, A. Voigt, and M. Hunziker. Linking demand and supply factors in identifying cultural ecosystem services of urban green infrastructures: A review of European studies. *Urban Forestry & Urban Greening*, 21(November):48–59, 2017. doi:10.1016/j.ufug.2016.11.002.

[HF19] J.-H. Haunert and A. Forsch. Inferring Personalized Multi-criteria Routing Models from Sparse Sets of Voluntarily Contributed Trajectories, 2019. `https://www.vgiscience.org/projects/vgi-routing.html`. DFG Priority Programme "Volunteered Geographic Information: Interpretation, Visualisation and Social Computing" (SPP 1894).

[HGM02] G. Hake, D. Grünreich, and L. Meng. *Kartographie*. Walter de Gruyter, 8th edition, 2002.

[HK15] C. Haaland and C. Konijnendijk van den Bosch. Urban Forestry & Urban Greening Challenges and strategies for urban green-space planning in

cities undergoing densification: A review. *Urban Forestry & Urban Greening*, 14(4):760–771, 2015. doi:10.1016/j.ufug.2015.07.009.

[HLO12] B. Han, L. Liu, and E. Omiecinski. NEAT: Road Network Aware Trajectory Clustering. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 142–151, 2012. doi:10.1109/ICDCS.2012.31.

[HM16] J.-H. Haunert and W. Meulemans. Partitioning polygons via graph augmentation. In *Proc. 9th International Conference on Geographic Information Science (GIScience '16)*, pages 18–33. Springer, 2016. doi:10.1007/978-3-319-45738-3_2.

[Hoj96] M. Hojati. Optimal political districting. *Computers & Operations Research*, 23(12):1147–1161, 1996. doi:10.1016/S0305-0548(96)00029-9.

[HRR+07] D. H. Huson, D. C. Richter, C. Rausch, T. Dezulian, M. Franz, and R. Rupp. Dendroscope: An interactive viewer for large phylogenetic trees. *BMC Bioinformatics*, 8(1):1–6, 2007. doi:10.1186/1471-2105-8-460.

[HS71] S. W. Hess and S. 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.

[HS11] J.-H. Haunert and L. Sering. Drawing road networks with focus regions. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2555–2562, 2011.

[HSS+12] D. Haase, N. Schwarz, M. Strohbach, F. Kroll, and R. Seppelt. Synergies, Trade-offs, and Losses of Ecosystem Services in Urban Regions: an Integrated Multiscale Framework Applied to the Leipzig-Halle Region, Germany. *Ecology and Society*, 17(3), 2012. doi:10.5751/ES-04853-170322.

[HW07] L. Harrie and R. Weibel. Modelling the overall process of generalisation. In W. Mackaness, A. Ruas, and L. T. Sarjakoski, editors, *Generalisation of Geographic Information: Cartographic Modelling and Applications*, chapter 4, pages 67–87. Elsevier, 2007. doi:10.1016/B978-008045374-3/50006-5.

[HW10a] J.-H. Haunert and A. Wolff. Area aggregation in map generalisation by mixed-integer programming. *International Journal of Geographical Information Science*, 24(12):1871–1897, 2010. doi:10.1080/13658810903401008.

[HW10b] J.-H. Haunert and A. Wolff. Optimal and topologically safe simplification of building footprints. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '10)*, page 192, 2010. doi:10.1145/1869790.1869819.

[HWS+65] S. W. Hess, J. B. Weaver, H. J. Siegfeldt, J. N. Whelan, and P. A. Zitlau. Nonpartisan Political Redistricting by Computer. *Operations Research*, 13(6):998–1006, 1965. doi:10.1287/opre.13.6.998.

[ILGZ14] S. Indermühle, P. Laube, M. Geilhausen, and T. Zwicker. Multi-criteria aggregation for sensistive parcel-based census data. In *Proc. 8th International Conference on Geographic Information Science (GIScience '14)*, pages 306–310, 2014.

[Jai10] A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 3(8):651–666, 2010. doi:10.1016/j.patrec.2009.09.011.

[JC04] B. Jiang and C. Claramunt. A structural approach to the model generalization of an urban street network. *Geoinformatica*, 8(2):157–171, 2004. doi:10.1023/B:GEIN.0000017746.44824.70.

[JRT95] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 20:111–152, 1995. doi:10.1090/dimacs/020/02.

[KAB+10] J. Kopf, M. Agrawala, D. Bargeron, D. Salesin, and M. Cohen. Automatic generation of destination maps. *ACM Transactions on Graphics*, 29(6):158:1–158:12, dec 2010. doi:10.1145/1882261.1866184.

[Kad10] M. Kada. Aggregation of 3D Buildings with a Hybrid Data Model. In *ISPRS Archives – Volume XXXVIII Part 4, Geospatial Data and Geovisualization: Environment, Security, and Society, Special Joint Symposium of ISPRS Commission IV and AutoCarto 2010 in conjunction with ASPRS, Orlando, Florida, USA, November 15-19*, 2010.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972. doi:10.1007/978-1-4684-2001-2_9.

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

[KCS04] M.-P. Kwan, I. Casas, and B. Schmitz. Protection of Geoprivacy and Accuracy of Spatial Information: How Effective Are Geographical Masks? *Cartographica: The International Journal for Geographic Information and Geovisualization*, 39(2):15–28, 2004. doi:10.3138/X204-4223-57MK-8273.

[Kes13] F. Kessler. Volunteered geographic information: A bicycling enthusiast perspective. *Cartography and Geographic Information Science*, 38(3):258–268, 2013. doi:10.1559/15230406382258.

[Kon94] I. Kononenko. Estimating attributes: analysis and extensions of RELIEF. In *European conference on machine learning*, pages 171–182. Springer, 1994. doi:10.1007/3-540-57868-4_57.

[KRA+14] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady. DBSCAN: Past, Present and Future. In *The 5th International Conference on the Applications of Digital Information and Web Technologies (ICADIWT 2014)*, pages 232–238, 2014. doi:10.1109/ICADIWT.2014.6814687.

[KRS10] H. Kriegel, M. Renz, and M. Schubert. Route skyline queries: A multi-preference path planning approach. In *Proc. 26th International Conference on Data Engineering (ICDE '10)*, pages 261–272, 2010. doi:10.1109/ICDE.2010.5447845.

[KYN07] F. Kong, H. Yin, and N. Nakagoshi. Using GIS and landscape metrics in the hedonic price modeling of the amenity value of urban green space: A case study in Jinan City, China. *Landscape and Urban Planning*, 79(3-4):240–252, 2007. doi:10.1016/j.landurbplan.2006.02.013.

[Lap92] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, Jun 1992. doi:10.1016/0377-2217(92)90138-Y.

[LCG14] W. Li, R. L. Church, and M. F. Goodchild. The p-Compact-regions Problem. *Geographical Analysis*, 46(3):250–273, Jul 2014. doi:10.1111/gean.12038.

[LGC13] W. Li, M. F. Goodchild, and R. L. Church. An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems. *International Journal of Geographical Information Science*, 27(6):1227–1250, Jun 2013. doi:10.1080/13658816.2012.752093.

[LHW07] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory Clustering: A Partition-and-Group Framework. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 593–604. Association for Computing Machinery, 2007. doi:10.1145/1247480.1247546.

[Llo82] S. P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

[LM10] A. C. K. Lee and R. Maheswaran. The health benefits of urban green spaces: a review of the evidence. *Journal of Public Health*, 33(2):212–222, 2010. doi:10.1093/pubmed/fdq068.

[LO93] Z. Li and S. Openshaw. A Natural Principle for the Objective Generalization of Digital Maps. *Cartography and Geographic Information Systems*, 20(1):19–29, 1993. doi:10.1559/152304093782616779.

[LR05] J. T. Linderoth and T. K. Ralphs. Noncommercial Software for Mixed-Integer Linear Programming. *Integer Programming: Theory and Practice*, 3:253–303, 2005. doi:10.1201/9781420039597.ch10.

[LRP95] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In

Proc. ACM Conference on Human Factors in Computing Systems (SIGCHI'95), pages 401–408. ACM Press/Addison-Wesley Publishing Co., 1995. doi:10.1145/223904.223956.

[LYT13] P. M. Lerin, D. Yamamoto, and N. Takahashi. Encoding network-constrained travel trajectories using routing algorithms. *International Journal of Knowledge and Web Intelligence*, 4(1):34–49, 2013. doi:10.1504/IJKWI.2013.052724.

[LZCJ08] A. Ligmann-Zielinska, R. L. Church, and P. Jankowski. Spatial optimization as a generative technique for sustainable multiobjective land-use allocation. *International Journal of Geographical Information Science*, 22(6):601–622, 2008. doi:10.1080/13658810701587495.

[Mac85] A. M. MacEachren. Compactness of Geographic Shape: Comparison and Evaluation of Measures. *Geografiska Annaler. Series B, Human Geography*, 67(1):53, 1985. doi:10.2307/490799.

[Mac07] W. A. Mackaness. Understanding geographic space. In *Generalisation of Geographic Information: Cartographic Modelling and Applications*, pages 1–10. Elsevier, 2007. doi:10.1016/B978-008045374-3/50003-X.

[Maf03] F. Maffray. On the coloration of perfect graphs. In B. A. Reed and C. L. Linhares-Sales, editors, *Recent Advances in Algorithms and Combinatorics*, CMS Books in Mathematics / Ouvrages de mathématiques de la SMC, pages 65–84. Springer New York, 2003. doi:10.1007/0-387-22444-0_3.

[MB93] W. A. Mackaness and M. K. Beard. Use of graph theory to support map generalization. *Cartography and Geographic Information Systems*, 20:210–221, 1993. doi:10.1559/152304093782637479.

[MCVL02] L. Muyldermans, D. Cattrysse, D. Van Oudheusden, and T. Lotan. Districting for Salt Spreading Operations. *European Journal of Operational Research*, 139(3):521–532, 2002. doi:10.1016/S0377-2217(01)00184-9.

[MGK+17] U. Mörtberg, R. Goldenberg, Z. Kalantari, O. Kordas, B. Deal, B. Balfors, and V. Cvetkovic. Integrating ecosystem services in the assessment of urban energy trajectories – A study of the Stockholm Region. *Energy Policy*, 100:338–349, 2017. doi:10.1016/j.enpol.2016.09.031.

[Mit02] J. E. Mitchell. Branch-and-Cut Algorithms for Combinatorial Optimization Problems. *Handbook of Applied Optimization*, pages 65–77, 2002.

[MJN98] A. Mehrotra, E. L. Johnson, and G. L. Nemhauser. An Optimization Based Heuristic for Political Districting. *Management Science*, 44(8):1100–1114, 1998. doi:10.1287/mnsc.44.8.1100.

[MM99] W. A. Mackaness and G. A. Mackechnie. Automating the Detection and Simplification of Junctions in Road Networks. *GeoInformatica*, 3(2):185–200, 1999. doi:10.1023/A:1009807927991.

[MNV09] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The Planar k-means Problem is NP-hard. In *WALCOM: Algorithms and Computation*, pages 274–285. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-00202-1_24.

[MRTH10] G. R. McCormack, M. Rock, A. M. Toohey, and D. Hignell. Health & Place Characteristics of urban parks associated with park use and physical activity: A review of qualitative research. *Health & Place*, 16(4):712–726, 2010. doi:10.1016/j.healthplace.2010.03.003.

[MS92] R. B. McMaster and K. S. Shea. *Generalization in Digital Cartography*. Association of American Geographers, Washington, D. C., 1992.

[MS01] M. Meilă and J. Shi. A Random Walks View of Spectral Segmentation algorithm 4 Stochastic matrices with piecewise. In *Proceedings of AI and STATISTICS (AISTATS) 2001*, 2001.

[MT16] J. D. Mazimpaka and S. Timpf. Trajectory data mining: A review of methods and applications. *Journal of Spatial Information Science*, 2016(13):61–99, 2016. doi:10.5311/JOSIS.2016.13.263.

[Mun57] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. doi:10.1137/0105003.

[Nag65] S. S. Nagel. Simplified Bipartisan Computer Redistricting. *Stanford Law Review*, 17(5):863–899, 1965. doi:10.2307/1226994.

[Nat19] National Recreation and Park Association17:. NRPA Agency Performance Review, 2019. `https://www.nrpa.org/siteassets/nrpa-agency-performance-review.pdf`. accessed on 2019-07-16.

[Nav16] B. Naveh. JGraphT, 2016. `http://jgrapht.org/`.

[ND11] S. Nadi and M. R. Delavar. Multi-criteria, personalized route planning using quantifier-guided ordered weighted averaging operators. *International Journal of Applied Earth Observations and Geoinformation*, 13(3):322–335, 2011. doi:10.1016/j.jag.2011.01.003.

[Nic01] S. Nicholls. Measuring the accessibility and equity of public parks: a case study using GIS. *Managing Leisure*, 6(4):201–219, 2001. doi:10.1080/13606710110084651.

[NJW02] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[NNR+06] G. J. Norman, S. K. Nutter, S. Ryan, J. F. Sallis, K. J. Calfas, and K. Patrick. Community Design and Access to Recreational Facilities as Correlates of Adolescent Physical Activity and Body-Mass Index. *Journal of Physical Activity and Health*, 3:118–128, 2006. doi:10.1123/jpah.3.s1.s118.

[NOLH18] B. Niedermann, J. Oehrlein, S. Lautenbach, and J.-H. Haunert. A network flow model for the analysis of green spaces in urban areas. In *Proc. 10th International Conference on Geographic Information Science (GIScience '18)*, volume 114 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:16, 2018. doi:10.4230/LIPIcs.GISCIENCE.2018.13.

[NW88] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 1988. doi:10.1002/9781118627372.

[NW10] M. Nöllenburg and A. Wolff. Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming. *IEEE Transactions on Visualization and Computer Graphics*, 17(5):626–641, 2010. doi:10.1109/TVCG.2010.81.

[OFS+18] J. Oehrlein, A. Förster, D. Schunck, Y. Dehbi, R. Roscher, and J.-H. Haunert. Inferring routing preferences of bicyclists from sparse sets of trajectories. In *Proc. 3rd International Conference on Smart Data and Smart Cities*, volume IV-4/W7 of *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 107–114, 2018. doi:10.5194/isprs-annals-IV-4-W7-107-2018.

[OH17] J. Oehrlein and J.-H. 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.

[OJ07] K. Oh and S. Jeong. Assessing the spatial distribution of urban parks using GIS. *Landscape and Urban Planning*, 82(January):25–32, 2007. doi:10.1016/j.landurbplan.2007.01.014.

[ONH17] J. Oehrlein, B. Niedermann, and J.-H. Haunert. Inferring the parametric weight of a bicriteria routing model from trajectories. In *Proc. 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL '17)*, pages 59:1–59:4, 2017. doi:10.1145/3139958.3140033.

[ONH19] J. Oehrlein, B. Niedermann, and J.-H. Haunert. Analyzing the Supply and Detecting Spatial Patterns of Urban Green Spaces via Optimization. *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 87(4):137–158, 2019. doi:10.1007/s41064-019-00081-0.

[Ope77] S. Openshaw. A Geographical Solution to Scale and Aggregation Problems in Region-Building, Partitioning and Spatial Modelling. *Transactions of the Institute of British Geographers*, 2(4):459–472, 1977. doi:10.2307/622300.

[Ope84] S. Openshaw. The modifiable areal unit problem. *Concepts and Techniques in Modern Geography*, 38:1–41, 1984.

[Orl13] J. B. Orlin. Max flows in O(nm) time, or better. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing - STOC '13*, pages 0–22, Palo Alto, 2013. Association for Computing Machinery. doi:10.1145/2488608.2488705.

[OvDH17] J. Oehrlein, T. C. van Dijk, and J.-H. Haunert. Gleichwertige Ziele in dynamischen Navigationskarten. In *DGPF Tagungsband*, volume 26, pages 138–146, 2017.

[OW02] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, Berlin, Heidelberg, 4th edition, 2002. doi:10.1007/978-3-8274-2804-2.

[Pat03] G. Pataki. Teaching Integer Programming Formulations Using the Traveling Salesman Problem. *SIAM Review*, 45(1):116–123, 2003. doi:10.1137/S00361445023685.

[PCC+11] D. E. Pataki, M. M. Carreiro, J. Cherrier, N. E. Grulke, V. Jennings, S. Pincetl, R. V. Pouyat, T. H. Whitlow, and W. C. Zipperer. Coupling biogeochemical cycles in urban environments: Ecosystem services, green solutions, and misconceptions. *Frontiers in Ecology and the Environment*, 9(1):27–36, 2011. doi:10.1890/090220.

[PN02] K. L. Papps and J. O. Newell. Identifying Functional Labour Market Areas in New Zealand: A Reconnaissance Study Using Travel-to-Work Data. *IZA Discussion Papers*, 443(1), 2002. `http://ftp.iza.org/dp443.pdf`.

[PPP+09] M. L. Potestio, A. B. Patel, C. D. Powell, D. A. McNeil, R. D. Jacobson, and L. McLaren. Is there an association between spatial access to parks/green space and childhood overweight/obesity in Calgary, Canada? *International Journal of Behavioral Nutrition and Physical Activity*, 6(1):77, 2009. doi:10.1186/1479-5868-6-77.

[PR91] M. W. Padberg and G. Rinaldi. A Branch-And-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. *SIAM Review*, 33(1):60–100, 1991. doi:10.1137/1033004.

[PR05] J. Puchinger and G. Raidl. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. In J. Mira and J. R. Álvarez, editors, *Artificial Intelligenceand Knowledge EngineeringApplications: A Bioinspired Approach*, pages 41–53. Springer, Berlin, Heidelberg, 2005. doi:10.1007/11499305_5.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Dover Publications, Inc., Mineola, NY, USA, 3rd edition, 1982.

[PT08] C. Puppe and A. Tasnádi. A computational approach to unbiased districting. *Mathematical and Computer Modelling*, 48(9–10):1455–1460, 2008. doi:10.1016/j.mcm.2008.05.024.

[PV13] T. E. Panduro and K. L. Veie. Landscape and Urban Planning Classification and valuation of urban green spaces—A hedonic house price valuation. *Landscape and Urban Planning*, 120:119–128, 2013. doi:10.1016/j.landurbplan.2013.08.009.

[RER+06] J. N. Roemmich, L. H. Epstein, S. Raja, L. Yin, J. Robinson, and D. Winiewicz. Association of access to parks and recreational facilities with the physical activity of young children. *Preventive Medicine*, 43:437–441, 2006. doi:10.1016/j.ypmed.2006.07.007.

[RM07] N. Regnauld and R. B. McMaster. A synoptic view of generalisation operators. In *Generalisation of Geographic Information: Cartographic Modelling and Applications*, pages 37–66. Elsevier, 2007. doi:10.1016/B978-008045374-3/50005-3.

[RS08] F. Ricca and B. 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.

[RSD+10] S. Reddy, K. Shilton, G. Denisov, C. Cenizal, D. Estrin, and M. B. Srivastava. Biketastic: sensing and mapping for better biking. In *Proc. 28th International Conference on Human Factors in Computing Systems (CHI '10)*, pages 1817–1820, 2010. doi:10.1145/1753326.1753598.

[RSS13] F. Ricca, A. Scozzari, and B. Simeone. Political Districting: From classical models to recent approaches. *Annals of Operations Research*, 204:271–299, 2013. doi:10.1007/s10479-012-1267-2.

[RU01] R. L. Rardin and R. Uzsoy. Experimental Evaluation of Heuristic Optimization Algorithms: A Tutorial. *Journal of Heuristics*, 7:261–304, 2001. doi:10.1023/A:1011319115230.

[SAM06] U. Sandström, P. Angelstam, and G. Mikusiński. Ecological diversity of birds in relation to the structure of urban green space. *Landscape and Urban Planning*, 77(1-2):39–53, 2006. doi:10.1016/j.landurbplan.2005.01.004.

[Sar07] L. T. Sarjakoski. Conceptual models of generalisation and multiple representation. In *Generalisation of Geographic Information: Cartographic Modelling and Applications*, pages 11–35. Elsevier, 2007. doi:10.1016/B978-008045374-3/50004-1.

[SBHD17] J. Sultan, G. Ben-Haim, J.-H. Haunert, and S. Dalyot. Extracting spatial patterns in bicycle routes from crowdsourced data. *Transactions in GIS*, 21(6):1321–1340, 2017.

[SBW06] S. Steiniger, D. Burghardt, and R. Weibel. Recognition of Island Structures for Map Generalization. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 67–74, 2006. doi:10.1145/1183471.1183484.

[Sch17] F. Schröter. Orientierungswerte (Richtwerte) für die Planung, 2017. `https://www.dr-frank-schroeter.de/planungsrichtwerte.htm`. accessed on 2019-07-16.

[Ses05] M. Sester. Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science*, 19(8–9):871–897, 2005. doi:10.1080/13658810500161179.

[She98] K. Sherstyuk. How to gerrymander: A formal analysis. *Public Choice*, 95(1–2):27–49, 1998. doi:10.1023/A:1004986314885.

[Shi05a] T. 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.

[Shi05b] T. Shirabe. Classification of Spatial Properties for Spatial Allocation Modeling. *GeoInformatica*, 9(3):269–287, 2005. doi:10.1007/s10707-005-1285-1.

[Shi09] T. Shirabe. Districting modeling with exact contiguity constraints. *Environment and Planning B: Planning and Design*, 36(6):1053–1066, 2009. doi:10.1068/b34104.

[SM00] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000. doi:10.1109/34.868688.

[Sma74] M. W. Smart. Labour market areas: Uses and definition. *Progress in Planning*, 2:239–353, 1974. doi:10.1016/0305-9006(74)90008-7.

[SS12] P. Sanders and D. Schultes. Engineering highway hierarchies. *ACM Journal of Experimental Algorithmics*, 17(1), 2012. doi:10.1145/2133803.2330080.

[ST02] T. Shirabe and C. D. Tomlin. Decomposing Integer Programming Models for Spatial Allocation. In M. J. Egenhofer and D. M. Mark, editors, *Geographic Information Science*, chapter 21, pages 300–312. Springer, Berlin, Germany, 2002. doi:10.1007/3-540-45799-2_21.

[ST04] D. A. Spielman and S.-H. Teng. Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time. *Journal of the ACM*, 51(3):385–463, 2004, 0111050. doi:10.1145/990308.990310.

[SWW10] C. Sister, J. R. Wolch, and J. Wilson. Got green? Addressing environmental justice in park provision. *GeoJournal*, 75(3):229–248, 2010. doi:10.1007/s10708-009-9303-8.

[Tal13] E. Talen. The social equity of urban service distribution: An exploration of park access in Pueblo, Colorado, and Macon, Georgia. *Urban Geography*, 18(6):521–541, 2013. doi:10.2747/0272-3638.18.6.521.

[TB02] R. C. Thomson and R. Brooks. Exploiting perceptual grouping for map analysis, understanding and generalization: The case of road and river networks. In D. Blostein and Y.-B. Kwon, editors, *Graphics Recognition Algorithms and Applications, GREC 2001*, volume 2390 of *Lecture Notes in Computer Science*, pages 148–157. Springer, 2002. doi:10.1007/3-540-45868-9_12.

[TES+11] M. Toftager, O. Ekholm, J. Schipperijn, U. Stigsdotter, P. Bentsen, M. Grønbæk, T. B. Randrup, and F. Kamper-Jørgensen. Distance to Green Space and Physical Activity: A Danish National Representative Survey. *Journal of Physical Activity and Health*, pages 741–749, 2011. doi:10.1123/jpah.8.6.741.

[TMS07] L. Tyrväinen, K. Mäkinen, and J. Schipperijn. Tools for mapping social values of urban woodlands and other green areas. *Landscape and Urban Planning*, 79(1):5–19, 2007. doi:10.1016/j.landurbplan.2006.03.003.

[TP66] F. Töpfer and W. Pillewizer. The principles of selection. *The Cartographic Journal*, 3(1):10–16, 1966. doi:10.1179/caj.1966.3.1.10.

[TPSdV05] L. Tyrväinen, S. Pauleit, K. Seeland, and S. de Vries. Benefits and Uses of Urban Forests and Trees. In C. Konijnendijk, K. Nilsson, T. Randrup, and J. Schipperijn, editors, *Urban Forests and Trees: A Reference Book*, chapter 4, pages 81–114. Springer, Berlin, 2005. doi:10.1007/3-540-27684-X_5.

[TWKS98] A. F. Taylor, A. Wiley, F. E. Kuo, and W. C. Sullivan. Growing up in the inner city: Green spaces as places to grow. *Environment and Behavior*, 30(1):3–27, 1998. doi:10.1177/0013916598301001.

[vDH14] T. C. van Dijk and J.-H. Haunert. Interactive focus maps using least-squares optimization. *International Journal of Geographical Information Science*, 28(10):2052–2075, 2014. doi:10.1080/13658816.2014.887718.

[vDHO16] T. C. van Dijk, J.-H. Haunert, and J. Oehrlein. Location-dependent generalization of road networks based on equivalent destinations. *Computer Graphics Forum*, 35(3):451–460, 2016. doi:10.1111/cgf.12921.

[vDvGH+13] T. C. van Dijk, A. van Goethem, J.-H. Haunert, W. Meulemans, and B. Speckmann. Accentuating focus maps via partial schematization. In *Proc. 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS '13)*, pages 428–431. ACM, 2013. doi:10.1145/2525314.2525452.

[VHW03] A. Van Herzele and T. Wiedemann. A monitoring tool for the provision of accessible and attractive urban green spaces. *Landscape and Urban Planningd*, 63(2):109–126, 2003. doi:10.1016/S0169-2046(02)00192-5.

[vO95] P. van Oosterom. The gap-tree, an approach to 'on-the-fly' map generalization of an area partitioning. *GIS and Generalization: Methodology and Practice*, pages 120–132, 1995.

[vRW87] T. J. van Roy and L. A. Wolsey. Solving Mixed Integer Programming Problems Using Automatic Reformulation. *Operations Research*, 35(1):45–57, 1987. doi:10.1287/opre.35.1.45.

[VvWvdL06] R. Vliegen, J. J. van Wijk, and E.-J. van der Linden. Visualizing business data with generalized treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):789–796, Sept 2006. doi:10.1109/TVCG.2006.200.

[War87] A. Warburton. Approximation of Pareto Optima in Multiple-Objective, Shortest-Path Problems. *Operations Research*, 35(1):70–79, 1987. doi:10.1287/opre.35.1.70.

[WBB17] Y. Wang, A. Buchanan, and S. Butenko. On imposing connectivity constraints in integer programs. *Mathematical Programming*, 166(1):241–271, Nov 2017. doi:10.1007/s10107-017-1117-8.

[WBN14] J. R. Wolch, J. Byrne, and J. P. Newell. Urban green space, public health, and environmental justice: The challenge of making cities 'just green enough'. *Landscape and Urban Planning*, 125(Supplement C):234–244, 2014. doi:10.1016/j.landurbplan.2014.01.017.

[Wei97] R. Weibel. Generalization of spatial data: Principles and selected algorithms. In *Algorithmic Foundations of Geographic Information Systems*, pages 99–152. Springer, Berlin, Germany, 1997. doi:10.1007/3-540-63818-0_5.

[Wil02] J. C. Williams. A Zero-One Programming Model for Contiguous Land Acquisition. *Geographical Analysis*, 34(4):330–349, 2002. doi:10.1111/j.1538-4632.2002.tb01093.x.

[WJB95] J. M. Ware, C. B. Jones, and G. L. Bundy. A Triangulated Spatial Model for Cartographic Generalisation of Areal Objects. In A. U. Frank and W. Kuhn, editors, *Spatial Information Theory A Theoretical Basis for GIS*, pages 173–192. Springer Berlin Heidelberg, 1995. doi:10.1007/3-540-60392-1_12.

[WJT03] J. M. Ware, C. B. Jones, and N. Thomas. Automated map generalization with multiple operators: a simulated annealing approach. *International Journal of Geographical Information Science*, 17(8):743–769, 2003. doi:10.1080/1365881031000159608 5.

[Wri42] J. K. Wright. Map makers are human: Comments on the subjective in maps. *Geographical review*, 32(4):527–544, 1942. doi:10.1002/9780470979587.ch40.

[XW05] R. Xu and D. Wunsch II. Survey of Clustering Algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005. doi:10.1109/TNN.2005.845141.

[Yag88] R. R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):183–190, 1988. doi:10.1109/21.87068.

[YLL11] B. Yang, X. Luan, and Q. Li. Generating hierarchical strokes from urban street networks based on spatial pattern recognition. *International Journal of Geographical Information Science*, 25(12):2025–2050, 2011. doi:10.1080/13658816.2011.570270.

[YOT09] D. Yamamoto, S. Ozeki, and N. Takahashi. Focus+glue+context: An improved fisheye approach for web map services. In *Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '09)*, pages 101–110, 2009. doi:10.1145/1653771.1653788.

[YZZ+10] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: Driving directions based on taxi trajectories. In *Proc. 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS '10)*, pages 99–108, 2010. doi:10.1145/1869790.1869807.

[ZE81] S. H. Zanakis and J. R. Evans. Heuristic "Optimization": why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981. doi:10.1287/inte.11.5.84.

[Zha05] Q. Zhang. Road network generalization based on connection analysis. In *Developments in Spatial Data Handling – Proc. 11th Internat. Sympos. Spatial Data Handling (SDH '04)*, pages 343–353. Springer-Verlag, Berlin, Germany, 2005. doi:10.1007/3-540-26772-7_26.

[ZHT06] Z. Zhang, K. Huang, and T. Tan. Comparison of Similarity Measures for Trajectory Clustering in Outdoor Surveillance Scenes. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 1135–1138, 2006. doi:10.1109/ICPR.2006.392.

[ZR02] A. Zipf and K.-F. Richter. Using focus maps to ease map reading: Developing smart applications for mobile devices. *Künstliche Intelligenz*, 02(4):35–37, 2002.

[ZS83] A. A. Zoltners and P. Sinha. Sales Territory Alignment: A Review and Model. *Management Science*, 29(11):1237–1256, 1983. doi:10.1287/mnsc.29.11.1237.